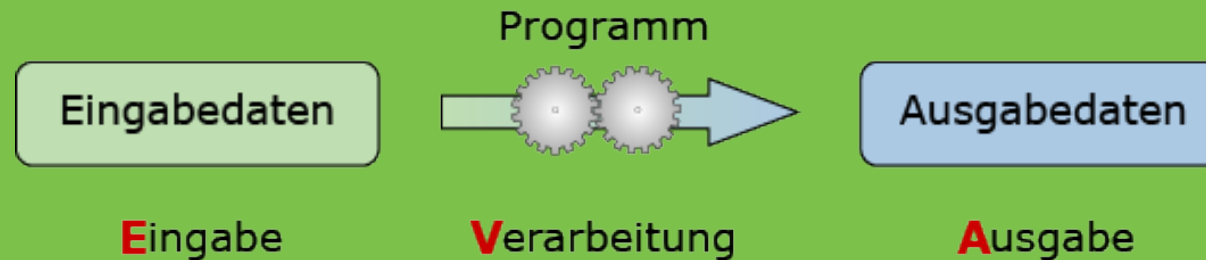




Programmierung 1

– Logik und Bedingungen



Yvonne Jung

Wiederholung



- Variablen dienen dazu Daten zu verwalten
 - Haben Bezeichner, Speicheradresse und Datentyp (z.B. `int`)
 - Variablenwerte können durch Zuweisungen verändert werden
- Aufbau von Zuweisungen: *<variable> = <ausdruck>;*
 - Wert des Ausdrucks wird mittels aktuellem Variablenzustand ermittelt
 - Variablen auf linker Seite der Zuweisung wird ermittelter Wert zugeordnet
 - Zuweisung (mit Zuweisungsoperator `=`) ist selbst Ausdruck
 - Mit Semikolon abgeschlossener Ausdruck ist Anweisung
- Programm ist geordnete Folge von Anweisungen
 - Besteht damit aus Anweisungssequenz
 - Anweisungsblöcke durch *geschweifte Klammern* zusammengefasst

Anweisungssequenz

Anweisung
...
Anweisung

Boolesche Variablen

- Kennen Variablen vom Typ `int` (Ganzzahlen)
 - "`int`" ist ein sogenannter (primitiver) Datentyp
- Ein weiterer Datentyp ist `bool`
 - Erst mit C99-Standard als "`_Bool`" eingeführt
 - Bindet man `stdbool.h` ein via: `#include <stdbool.h>` dann kann man, wie in C++, den Typ `bool` nehmen
- Variablen vom Typ `bool` können einen von zwei Werten annehmen
 - `false` (in C wird außerdem Zahlenwert `0` auch als `false` ausgewertet)
 - `true` (in C wird alles `ungleich 0` – also u.a. `1` – als `true` ausgewertet)
- Beispiel: `bool humanBeing = true;`

Programmiersprache C ist im wesentlichen Teilmenge von C++

Boolesche Ausdrücke



- Kennen bereits arithmetische Operatoren auf Typ `int`
- *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 1



- 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)
 - Auch Boolesche Werte können auf Gleichheit bzw. Ungleichheit getestet werden
 - >, >=, <, <= (größer/ größer gleich, kleiner/ kleiner gleich)
- In der **Bedingung** von Fallunterscheidungen (sog. "if"-Anweisungen) steht ein Ausdruck, der einen Wahrheitswert ergibt
 - *Fallunterscheidungen* dienen zur Modellierung alternativer Programmabläufe
 - Beispiel (zweiseitige Fallunterscheidung):

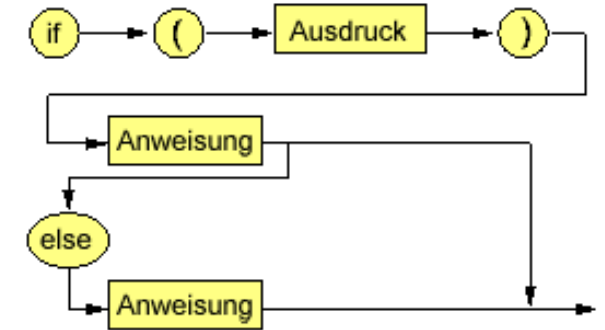
```
if (humanBeing == true) {  
    printf("Hallo Mensch!\n");  
}  
else {  
    printf("Hallo Alien!\n");  
}
```

 - Der "else"-Teil (grünlich unterlegt) ist optional
 - Nur "if"-Teil ohne "else"-Teil ist einseitige Fallunterscheidung

Bedingte Anweisungen

- Erst Bedingungsausdruck (in Klammern) auswerten
 - Z.B. *humanBeing == true* oder *n > 0*
 - Muss Wahrheitswert ergeben
- Falls *true*, weiter mit erstem Block, sonst mit zweitem
 - Bei einseitiger Fallunterscheidung gibt es nur ersten Block
 - Bsp.:

```
if (n > 0) {  
    printf("Die Zahl %d ist positiv.\n", n);  
}
```
- Geschweifte Klammern definieren Blöcke
 - Blöcke ordentlich einrücken (z.B. um 4 Leerzeichen)
- In Blöcken können wieder Fallunterscheidungen stehen
 - Es gibt auch leere Blöcke: `{ }`
- Schlüsselwörter: `if` und `else`



zweiseitige Fallunterscheidung

Bedingung	
w	f
Anweisung	Anweisung
...	...
Anweisung	Anweisung

Eingabe

```
int age;  
printf("Bitte Alter eingeben: ");  
scanf("%d", &age);  
printf("Du wurdest %d geboren.\n", 2021-age);
```

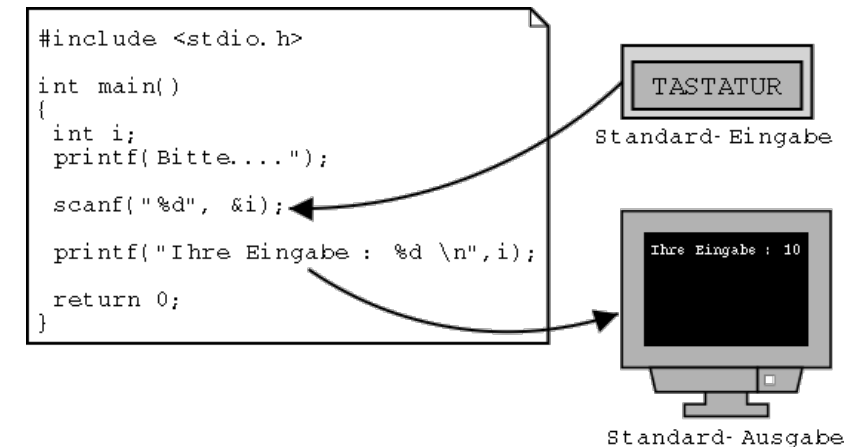
Variablenname	Datentyp	Speicheradresse	Speicherinhalt
age	int	48012	?

↑
&age

- Funktion *scanf()* liest von Standardeingabe

- *scanf()* ähnlich aufgebaut wie *printf()*
- Formatzeichen **%d** spezifiziert, dass ganze Zahl eingelesen werden soll
 - Einlesen bis inklusive letzter Ziffer
- Zeichen **&** ist der Adressoperator

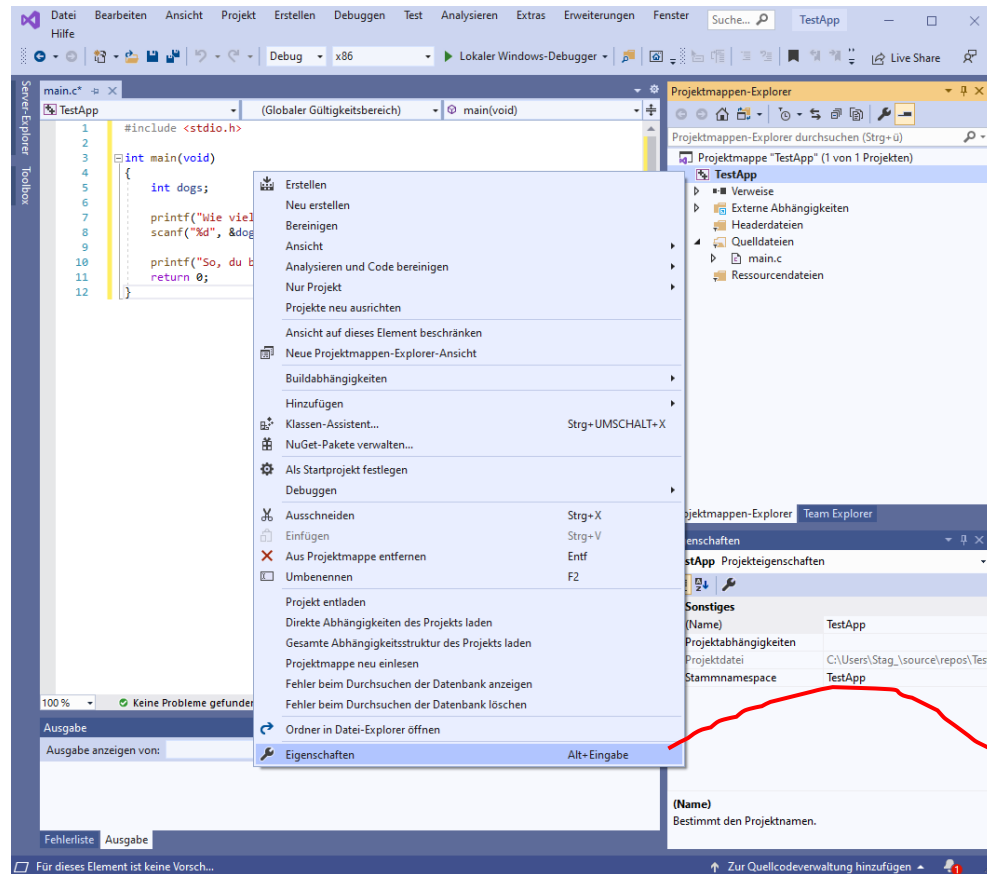
- In C muss man beim Einlesen von Werten Speicheradresse angeben (mittels Adressoperator), wo Wert im Speicher abgelegt werden soll





- Eingabefunktionen wie *scanf()* haben generell Sicherheitsprobleme
 - Leider auf jeder Plattform... Ist aber erst für Produktivcode relevant...
 - *scanf()* prüft nicht auf Pufferüberlauf, was unbehandelt Hackern Tür und Tor öffnet; nichtsdestotrotz ist es leicht zu verwenden und daher für den Einstieg einfacher zu nutzen als direkt die komplette Problembehandlung umzusetzen
- MS Visual Studio sieht solche Funktionen als veraltet (deprecated)
 - Man muss diese Funktionen explizit erlauben, s. Abb. auf nächster Folie: Projekteigenschaften öffnen und dort SDL-Prüfungen deaktivieren
 - Unter Linux und Mac zum Glück unnötig 😊
 - Alternativ gibt es seit C11-Standard (noch nicht überall unterstützt) eine sichere Variante davon (hinten "_s" anhängen), also hier: *scanf_s()*

Hochschule Fulda
University of Applied Sciences



Bedingungen – Beispiel

- Bestimmung des Maximums (also des größten Werts)

```
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 Bedingung $a > b$) und das Ergebnis wird x zugewiesen (falls Bedingung wahr ist, bekommt Variable x den Wert von a zugewiesen, sonst Wert von b)

Ternärer Operator



- Bedingungsoperator **?:** ist einziger ternärer Operator
 - D.h. Operator mit 3 Operanden (z.B. + ist binärer Operator)
- Syntax: *<Bedingungsausdruck> ? <Anweisung 1> : <Anweisung 2>*
 - Falls Bedingung wahr ist, wird erste Anweisung ausgewertet, sonst zweite
 - Nur ein Ausdruck wird je ausgewertet, und zwar der, für den die Bedingung zutrifft
 - Sozusagen Kurzform der "if/else"-Anweisung (vor allem bei Zuweisungen)
 - Ternärer Operator gilt für jeden Datentyp
 - Kann zugewiesen werden, wie andere Ausdrücke auch, oder allein stehen
- Beispiel zum Knobeln:

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

 - Hinweis für Interessierte: mit Platzhalter **%s** kann man ganze *Zeichenketten* ausgeben (d.h. in Anführungszeichen stehende Zeichenfolgen wie z.B. "ungerade")

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'` oder `'\t'`) aus ASCII-Zeichensatz
 - ASCII-Tabelle: <https://www.c-howto.de/tutorial/anhang/ascii-tabelle/>
 - Man kann mit einem `char` aber auch kleine Ganzzahlen abspeichern
 - Die Größe eines `char`'s beträgt genau 1 Byte
 - Wertebereich des Datentyps geht damit von -128 bis +127 (bzw. von 0 bis 255)
 - Angabe eines Zeichenwertes geschieht mit einfachen Anführungszeichen
- Für Ein- und Ausgabe eines Zeichens gibt es den Platzhalter **`%c`**
 - Beispiel: `char c0 = 'p', c1 = 'r', c2 = 'o', c3 = 'g', nl = '\n';`
`printf("%c%c%c%c%c", c0, c1, c2, c3, nl);`

Komplexere "if"-Anweisungen

- Ein einfacher Taschenrechner:

```
char operator;  
int wert1, wert2, erg;  
  
printf("Zu berechnenden 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);
```



```
if (operator == '+')  
    erg = wert1 + wert2;  
else if (operator == '-')  
    erg = wert1 - wert2;  
else if (operator == '*')  
    erg = wert1 * wert2;  
else if (operator == '/')  
    erg = wert1 / wert2;  
else if (operator == '%')  
    erg = wert1 % wert2;  
else  
    operator = ' ';
```

- Sollen für eine Bedingung mehrere Anweisungen ausgeführt werden, müssen diese mit geschweiften Klammern zu einem Block zusammengefasst werden

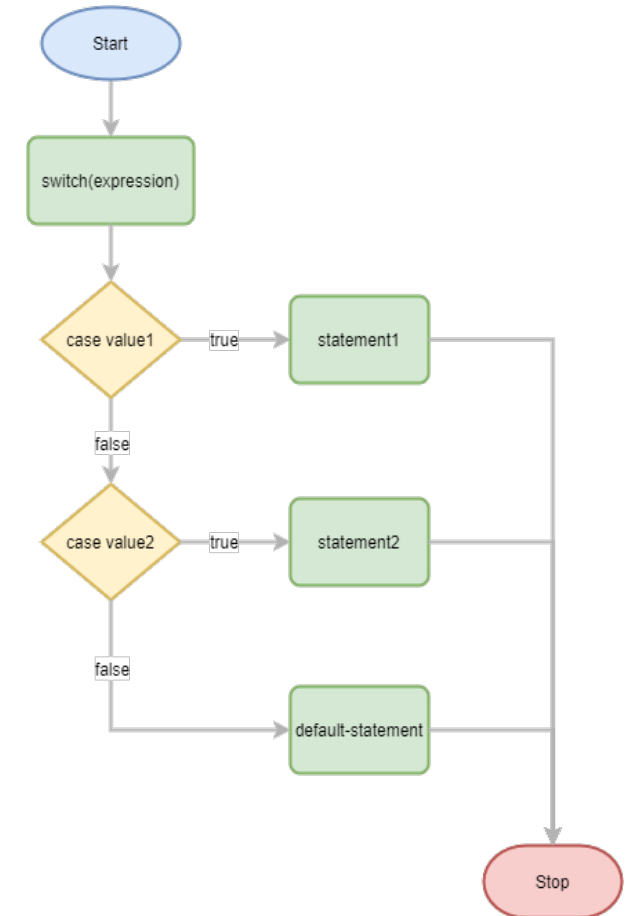
Mehrfachauswahl

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

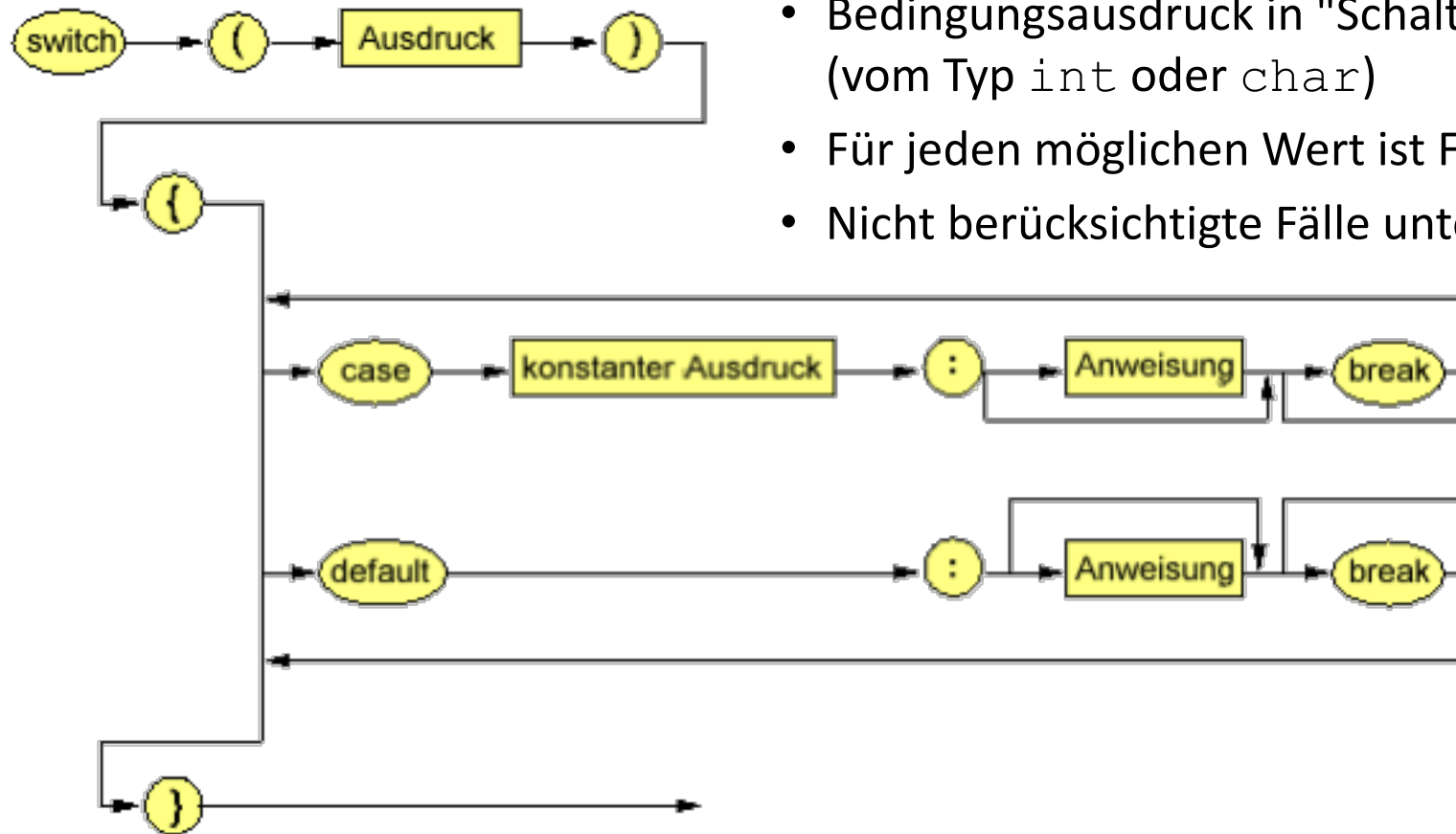
```
printf("Zu berechnenden Term angeben (z.B. 1 + 1)!\n");  
scanf("%d %c %d", &wert1, &operator, &wert2);
```

```
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 = ' ';  
}
```

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



Mehrfachauswahl

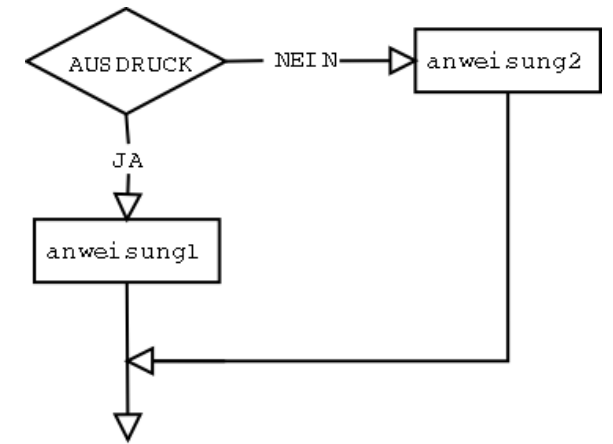


- Bedingungsausdruck in "Schalter" (`switch`) liefert Wert (vom Typ `int` oder `char`)
- Für jeden möglichen Wert ist Fall (`case`) vorgesehen
- Nicht berücksichtigte Fälle unter `default` behandeln

- Anweisung `break` bedeutet: hier wird abgebrochen und Ausführung erst nach `switch` fortgesetzt
- Ohne `break` wird je nachfolgendes `case` weiter abgearbeitet

Wrap-up

- Zwei Typen von Fallunterscheidungen
 - `if (Bedingung_erfuellt) { ... }`
`else if (andere_Bedingung) { ... }`
`else { ... }`
 - `switch(Ausdruck) { // Kann Typ char oder int sein`
`case Const_Term_1: ... break;`
`case Const_Term_2: ... break;`
`...`
`default: ...`
`}`
- Bedingungsoperator
 - `<Bedingung> ? <Anweisung 1> : <Anweisung 2>`



Achtung: Alles, was ungleich *false* bzw. *0* ist, gilt in C als wahr.

Logische Aussagen

- Als boolesche Operatoren haben wir *Und*, *Oder*, *Nicht* kennengelernt
 - Hinweis: je nach Fachrichtung gibt es dafür 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	$!a$	$\neg a$	\bar{a}

- In Digitaltechnik (und in C) wird **0** als *false* und **1** als *true* interpretiert
- Auswertungsreihenfolge
 - *Und* bindet stärker als *Oder*, aber *Nicht* bindet stärker als *Und* sowie *Oder*
 - Vergleichsoperatoren binden schwächer als *Nicht*, aber stärker als *Und* / *Oder*
 - Im Zweifel, oder möchte man andere Reihenfolge, (runde) Klammern setzen

Regeln



Kommutativgesetze

$$(1) \quad a \wedge b = b \wedge a$$

$$(1') \quad a \vee b = b \vee a$$

Assoziativgesetze

$$(2) \quad (a \wedge b) \wedge c = a \wedge (b \wedge c)$$

$$(2') \quad (a \vee b) \vee c = a \vee (b \vee c)$$

Idempotenzgesetze

$$(3) \quad a \wedge a = a$$

$$(3') \quad a \vee a = a$$

Distributivgesetze

$$(4) \quad a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$(4') \quad a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

Neutralitätsgesetze

$$(5) \quad a \wedge 1 = a$$

$$(5') \quad a \vee 0 = a$$

Extremalgesetze

$$(6) \quad a \wedge 0 = 0$$

$$(6') \quad a \vee 1 = 1$$

Doppelnegationsgesetz

$$(7) \quad \neg(\neg a) = a$$

De Morgansche Gesetze

$$(8) \quad \neg(a \wedge b) = \neg a \vee \neg b$$

$$(8') \quad \neg(a \vee b) = \neg a \wedge \neg b$$

Komplementärgesetze

$$(9) \quad a \wedge \neg a = 0$$

$$(9') \quad a \vee \neg a = 1$$

Dualitätsgesetze

$$(10) \quad \neg 0 = 1$$

$$(10') \quad \neg 1 = 0$$

Absorptionsgesetze

$$(11) \quad a \vee (a \wedge b) = a$$

$$(11') \quad a \wedge (a \vee b) = a$$

Übung 2



- Gegeben ist eine Funktion *main()*, 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 "true" ausgeben, wenn $a < b < c$, außer *ok* hat den Wert 'y', wobei dann die Bedingung $a < b$ nicht unbedingt gelten muss. Andernfalls soll "false" ausgegeben werden.
- Bitte Lückentext ausfüllen!

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

Übung 3



- Gegeben ist eine **main**-Funktion, die ein **int** namens *age* und ein **char** namens *stillGoingStrong* einliest. Gültige Werte für *stillGoingStrong* sind 'y' bzw. 'n' (wie bei Übung 2). Bei anderen Werten soll nur eine Fehlermeldung ausgegeben werden.
- Das Programm wertet ansonsten aus, ob eine Party mit Gästen vom Alter *age* erfolgreich ist. In diesem Fall soll "Erfolg" ausgegeben werden, sonst "Pleite".
- Eine Party ist erfolgreich, wenn das Alter zwischen 18 und 30 (je einschließlich) liegt. Die Party ist außerdem erfolgreich, falls *stillGoingStrong* den Wert 'y' hat (unabhängig vom Alter), denn in dem Fall haben auch noch Ältere Spaß daran.
- Genereller Hinweis: Ist bei *Oder* der erste Ausdruck wahr, wird der zweite nicht mehr ausgewertet. Ist analog bei *Und* der erste Ausdruck schon falsch, so wird der zweite auch nicht mehr ausgewertet.



Vielen Dank!

Noch Fragen?

