

PROGRAMMIEREN I

WS 2022

Prof. Dr.-Ing. Kolja Eger
Hochschule für Angewandte Wissenschaften Hamburg

Check-In



Präsenzklausur im Labor (180min)

Computertechnik

Praktikum 7

Projekte organisieren



Praktikum 6

Vektoren

Praktikum 5

Funktionen

Praktikum 4

Kontrollstrukturen



Praktikum 3

Datentypen & Operatoren



Praktikum 2

Erste Programme



Praktikum 1

Unser Weg durch das Semester

Themen

Vom letzten Mal

- Compiler, Linker, ..
- Datentypen
- Operatoren
- Kontrollstrukturen (Einstieg)
- Coding Style
- Beispiel: Fahrenheit -> Celsius

Von heute

- Typumwandlung
- Verkürzte Zuweisung
- Kontrollstrukturen

Wiederholung: Datentypen

Datentyp	Speicherbedarf	Kommentar
char	1 Byte	ganze Zahl oder ein Zeichen
short int	mind. 2 Byte	ganze Zahl
short	$\text{short} \leq \text{int}$	
int	$\text{short} \leq \text{int} \leq \text{long}$	ganze Zahl
long int	mind. 4 Byte	ganze Zahl
long	$\text{int} \leq \text{long}$	
float	$\text{float} \leq \text{double}$	Gleitkommazahl
double	$\text{float} \leq \text{double} \leq \text{long double}$	Gleitkommazahl
long double	$\text{double} \leq \text{long double}$	Gleitkommazahl

Typumwandlung

- Von welchem Typ ist das Ergebnis, falls ein Operator auf unterschiedliche Datentypen angewendet wird?
 - Beide Operanden werden verglichen und angepasst
 - Anpassung erfolgt nach Rang der Datentypen (siehe links)
 - Ein Operand auf niedrigerem Rang wird vor Anwendung des Operators auf den Rang des anderen Operanden umgewandelt
- Beispiel: 7-3ul
 - Links: Typ *int*
 - Rechts: Typ *unsigned long*
 - ➔ Vor der Operation wird 7 in *unsigned long* umgewandelt
 - ➔ Ergebnis ist auch vom Typ *unsigned long*
- *Char* und *short* werden mindestens in *int* umgewandelt
- Sonderfall: *unsigned int* und *long* (siehe Script)

Rang	Datentyp
1	long double
2	double
3	float
4	unsigned long
5*	long
6*	unsigned int
7	int

Typumwandlung bei Zuweisung

- Beispiel:

```
int i;  
char c = 65;
```

```
i = c;  
c = i;
```

```
printf("i=%d c=%d\n", i, c);
```

Wert von c
weiterhin 65

```
int i=12345;  
char c;
```

```
c = i;  
i = c;
```

```
printf("i=%d c=%d\n", i, c);
```

In diesem Fall
ändert sich
der Wert von i

- Bei einer Zuweisung wird der Datentyp der rechten Seite in den Datentyp der linken Seite umgewandelt!
- Bei einer Verkürzung kann es zu einer Warnung kommen
- Bei einer Verkürzung von einer Gleitkommazahl in eine ganze Zahl wird der Nachkommateil abgeschnitten
- Bei einer Reduzierung innerhalb der Gleitkommatypen ist das Verhalten systemabhängig (es wird abgeschnitten oder gerundet)

Explizite Typumwandlung (cast)

- Datentypen können explizit in einen anderen Datentypen umgewandelt werden
- Dies wird als *cast* bezeichnet
- Der Typ wird in runden Klammern vorgestellt
- Beim Kompilieren werden so Warnungen aufgelöst

```
int i=65;  
char c;  
  
c = (char) i;
```

 Cast – explizite Typumwandlung

→ Beispiel: Umrechnung Fahrenheit in Celsius

Version 1 mit int

```
19  /* Erstellung der Tabelle */
20  fahr = lower;
21  while (fahr <= upper) {
22      celsius = 5 * (fahr - 32) / 9;
23      printf("%3d\t%5.1f\n", fahr, celsius);
24      fahr = fahr + step;
25  }
```

Version 2 mit Typumwandlung

```
19  /* Erstellung der Tabelle */
20  fahr = lower;
21  while (fahr <= upper) {
22      celsius = 5 * ((float)fahr - 32) / 9;
23      printf("%3d\t%5.1f\n", fahr, celsius);
24      fahr = fahr + step;
25  }
```

Rangfolge der Operatoren

Rang	Operatoren	Reihenfolge
1	() [] -> .	von links nach rechts
2	! ~ ++ -- + - * & (type) sizeof	<i>von rechts nach links</i>
3	* / %	von links nach rechts
4	+ -	von links nach rechts
5	<< >>	von links nach rechts
6	< <= > >=	von links nach rechts
7	== !=	von links nach rechts
8	&	von links nach rechts
9	^	von links nach rechts
10		von links nach rechts
11	&&	von links nach rechts
12		von links nach rechts
13	?:	<i>von rechts nach links</i>
14	= += -= *= /= %= &= ^= = <<= >>=	<i>von rechts nach links</i>
15	,	von links nach rechts

Verkürzte Zuweisungen

- In der Regel sind Programmierer *faul*, zumindest gibt es in C Möglichkeiten schreib-faul zu werden
- Beispiel:

```
i = i + 3;
```

- Code besteht aus Ausdruck (i+3) und Zuweisung (i=)
- Dies kann in C verkürzt dargestellt werden

```
i += 3;
```

- Oder allgemein

Variable Operator= Ausdruck

Für die binären Operatoren +, -, *, /, %, &, |, ^, << und >>

- Aber Achtung:

```
a *= b+2;
```

ist

```
a = a*(b+2);
```

und nicht

```
a = a*b+2;
```

Weitere Zuweisungen

- .. und auch sowas ist in C erlaubt

```
a = (b=3)*4;
```

- Nach dieser Zeile ist b=3 und a=12
- Hier wird die Zuweisung (b=3) gleichzeitig als Ausdruck verwendet
- Verwenden Sie diese Verschachtelungen sehr sparsam. Es erhöht nicht die Lesbarkeit des Codes!

KONTROLLSTRUKTUREN

Kontrollstrukturen

- Ein Programm besteht aus Anweisungen
- Wenn Anweisungen nur bedingt oder mehrfach ausgeführt werden sollen, können Kontrollstrukturen genutzt werden, um dies zu steuern
- Kontrollstrukturen sind z.B.
 - Schleifen (*while, for, do*)
 - Verzweigungen (*if, if-else, switch-case*)
- Wenn ein Programm entsteht (oder Anforderungen für ein Programm spezifiziert werden) sind textuelle Beschreibungen nicht immer einfach zu verstehen oder eindeutig
- Zur Visualisierung werden deswegen häufig Diagramme verwendet (→ UML)

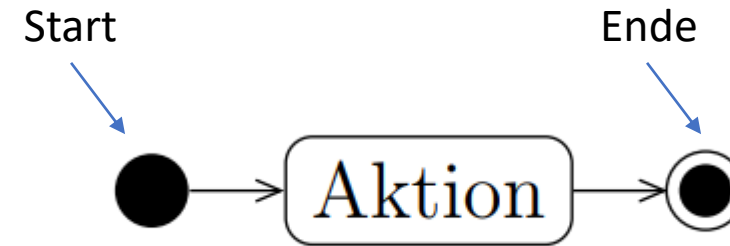
UML

UML (Unified Modeling Language) =
Sprache zur Beschreibung von Softwaresystemen

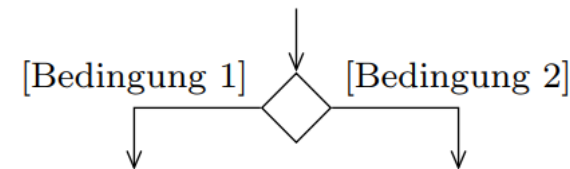
- UML ist ein Werkzeug für die Systemanalyse und beim Design → einheitliche Notation
- Abstrakte Beschreibungssprache ermöglicht Kommunikation zwischen Entwicklern und Benutzern
- **Verschiedene Diagrammtypen**, die sich gegenseitig **ergänzen** und verschiedene Systemaspekte **hervorheben**.
- UML umfasst:
 - Use-Case-Diagramme
 - Klassendiagramme
 - Interaktionsdiagramme
 - Package-Diagramme
 - Zustandsdiagramme
 - **Aktivitätsdiagramme**

Aktivitätsdiagramme

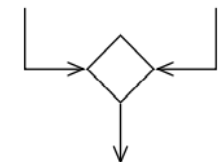
- Dies ist keine (vollständige) Einführung in UML
- Wir nutzen nur so viel, um Kontrollstrukturen in einem Diagramm zu beschreiben
- Es gibt weitere Elemente (z.B. für Parallelverarbeitung), die wir nicht näher betrachten
 - Im Internet gibt es viele Übersichten
 - Es gibt auch viele gute Bücher, z.B. „UML 2 glasklar“ von Rupp/Queins



- Aktionen/Aktivitäten werden als Rechteck mit abgerundeten Ecken dargestellt
- Start/Ende als Kreise
- Reihenfolge/Kontrollfluss wird durch Pfeile visualisiert

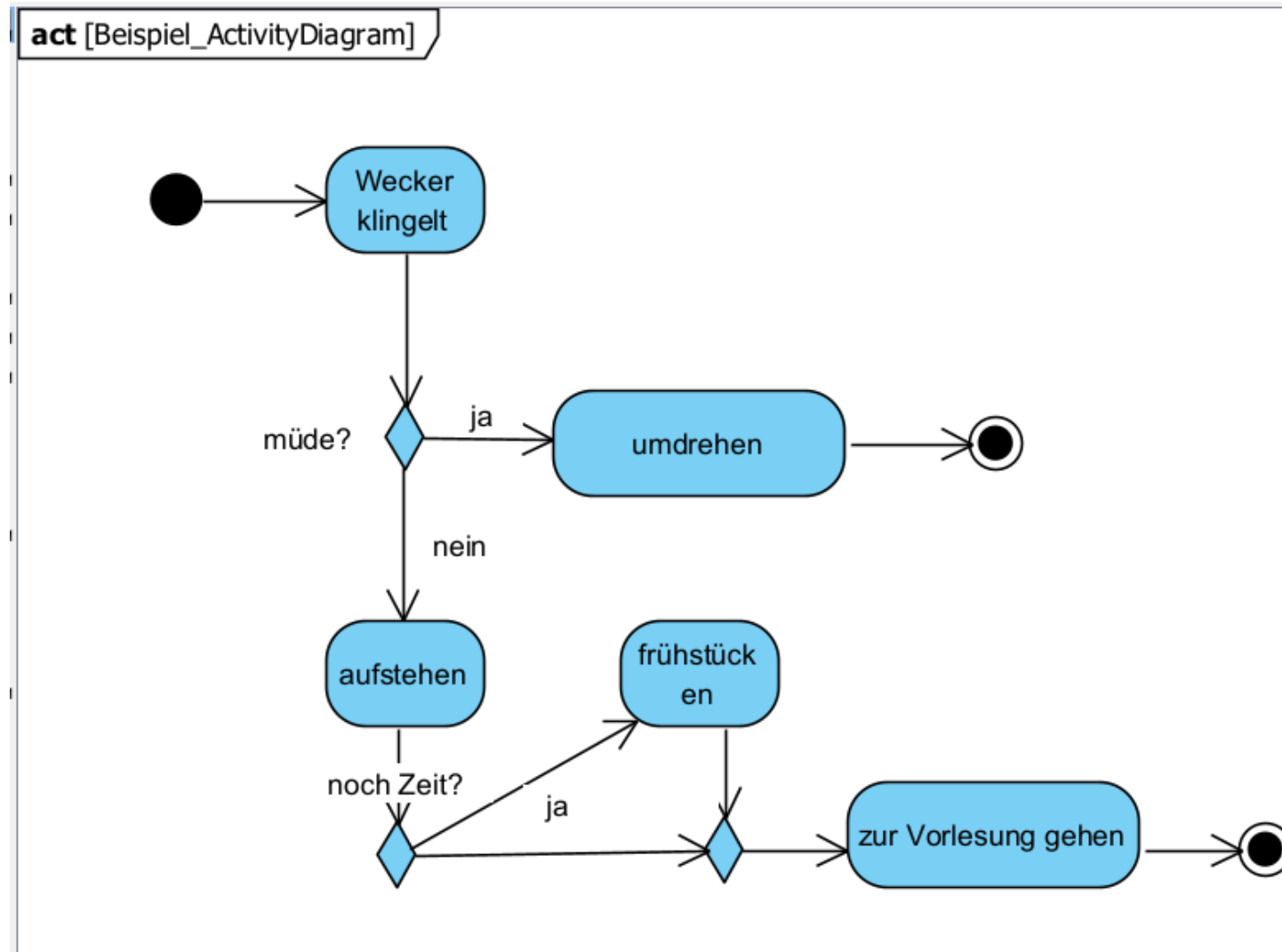


- Verzweigung (als Raute dargestellt)
- Bedingungen müssen vollständig & eindeutig sein



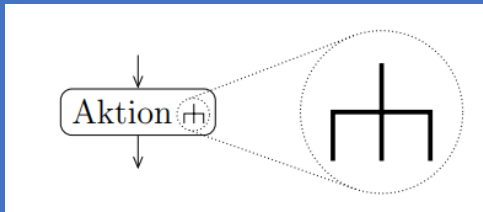
- Zusammenführung (auch als Raute)

Beispiel (erstellt mit Visual Paradigm Community Edition)



Aktivitätsdiagramme (II)

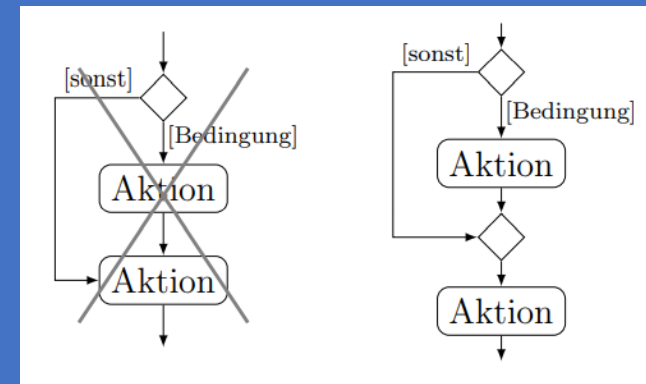
Aktivitätsdiagramme können verschachtelt werden („Gabel“-Symbol)



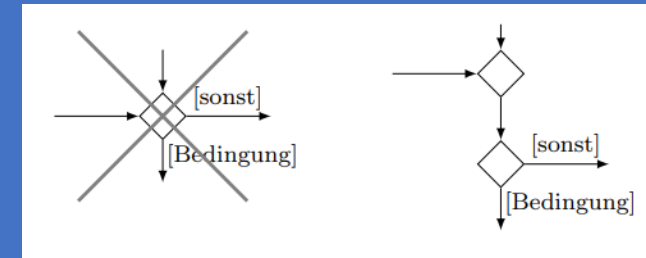
Aktivitätsdiagramme haben nur **einen Startknoten** aber evtl. **mehrere Ende**

Typische Fehler

Aktionen dürfen nicht gleichzeitig der Zusammenführung dienen



Zusammenführung und Verzweigung sollen getrennt werden



Schleifen

- for

for (Initialisierung; Schleifenbedingung; Änderung) Aktion

- while

while (Schleifenbedingung) Aktion

- do

do Aktion ***while*** (Schleifenbedingung)

Kopfgesteuerte Schleife
(*pre checked loop*):

Erst wird die Bedingung
geprüft, dann Aktion
ausgeführt

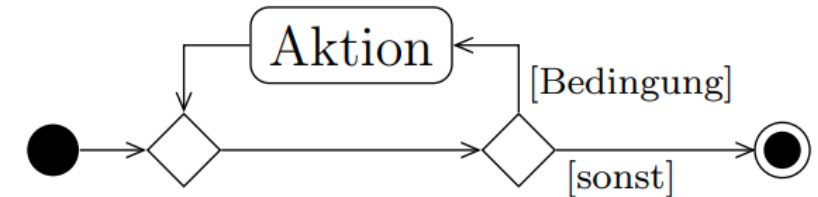
Fußgesteuerte Schleife
(*post checked loop*):

Bedingung wird erst
am Ende geprüft.

Schleife wird min.
einmal ausgeführt

While-Schleife

while (Schleifenbedingung) Aktion



Beispiel:

```
#include <stdio.h>

int main() {
    double Kontostand = 345.23;
    while (Kontostand >= 50) {
        Kontostand = Kontostand - 50;
        printf("50 Euro ausgegeben. Neuer Kontostand= %.2f\n", Kontostand);
    }
}
```

- Mit geschweiften Klammern werden in C Blöcke deklariert.
 - Hier werden somit mehrere Anweisungen in der Schleife durchgeführt
 - Zur besseren Lesbarkeit werden die Codezeilen innerhalb eines Blockes eingerückt
 - Ohne {} würde nur die erste Zeile nach dem while() ausgeführt werden
 - Alle Punkte gelten für alle Kontrollstrukturen (Ausnahme: switch)
-
- Achten Sie darauf, dass innerhalb der Schleife min. eine Variable der Schleifenbedingung verändert wird und somit die Schleife ein Ende findet

While-Schleife

- Was macht dieses Programm?

```
#include <stdio.h>
int main()
{
    int i = 0;
    while (1)
        printf("%d\n", i++);

    return 0;
}
```

For-Schleife

for (Initialisierung; Schleifenbedingung; Änderung) Aktion

Wie sieht das Aktivitäten-
diagramm für for-Schleife aus?

Beispiel:

```
int i = 0;
Kontostand = 345.23;
for (i = 0; i < 5; i++) {
    Kontostand = Kontostand - 50;
    printf("50 Euro ausgeben. Neuer Kontostand= %.2f\n", Kontostand);
}
```

- For-Schleife wird bei einfachen Bedingungen bevorzugt, z.B. x-mal ausführen oder *von/bis*

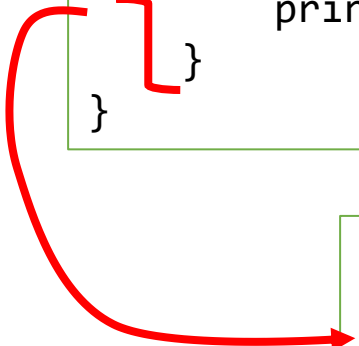
- Felder in der for-Schleife können auch weggelassen werden → hier z.B. `i=0`, da es schon vorher initialisiert wurde
- Leere Felder werden als „wahr“ angenommen
- Weglassen ist aber unüblich und erschwert die Lesbarkeit!

For-Schleife

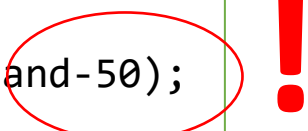
- Auch mit der *for*-Schleife lässt sich das Beispiel für *while()* implementieren

```
#include <stdio.h>

int main() {
    double Kontostand = 345.23;
    while (Kontostand >= 50) {
        Kontostand = Kontostand - 50;
        printf("50 Euro ausgegeben. Neuer Kontostand= %.2f\n", Kontostand);
    }
}
```

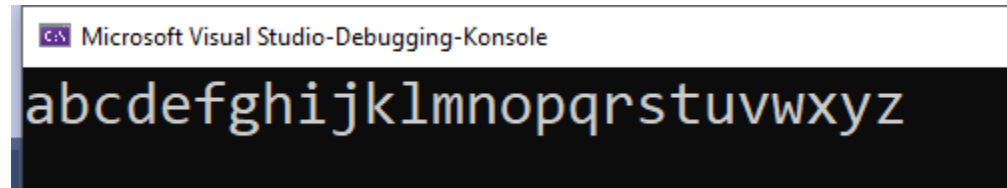


```
for (Kontostand = 345.23; Kontostand >= 50; Kontostand -= 50)
    printf("50 Euro ausgegeben. Neuer Kontostand= %.2f\n", Kontostand-50);
```



Übung

- Gebe Sie alle Buchstaben des Alphabets aus (mit einer Schleife!)

A screenshot of the Microsoft Visual Studio Debugging Console. The title bar at the top reads "Microsoft Visual Studio-Debugging-Konsole". The main area of the console is black with white text displaying the lowercase alphabet "abcdefghijklmnopqrstuvwxyz".

```
Microsoft Visual Studio-Debugging-Konsole  
abcdefghijklmnopqrstuvwxyz
```


Do-Schleife

do Aktion ***while*** (Schleifenbedingung)

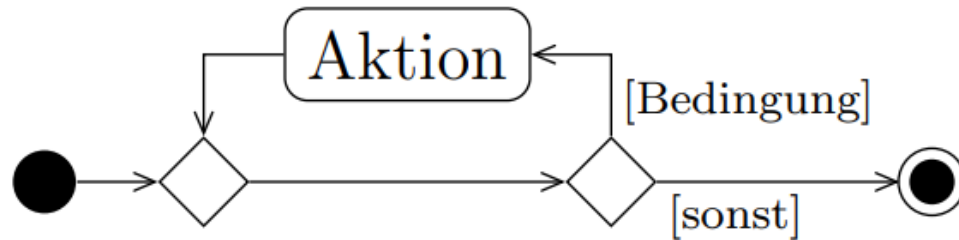
- Schleifenbedingung wird am Ende geprüft!
- Schleife wird min. einmal ausgeführt

Beispiel:

```
int Zahl;  
do {  
    printf("Geben Sie eine ganze Zahl groesser 0 ein:");  
    scanf("%d", &Zahl);  
} while (Zahl <= 0);
```

Übung zeichnen Sie das Aktivitätsdiagramm für eine do-Schleife (zu zweit!)

Aktivitätsdiagramm *for/while*



Aktivitätsdiagramm *do*

Unterbrechungen von Schleifen

(*break* und *continue*)

- Die bisherigen Schleifen prüfen die Bedingung am Anfang oder am Ende
- Um eine Schleife irgendwo in der Mitte zu beenden, können zusätzlich zwei Befehle verwendet werden
 - *break* – beendet die Schleife ohne erneutes Prüfen der Schleifenbedingung
 - *continue* – unterbricht den aktuellen Durchlauf innerhalb der Schleife und es folgt eine erneute Überprüfung der Schleifenbedingung
- Verwenden Sie beide Befehle sparsam!
 - Häufig ist die Lesbarkeit des Codes schlechter
 - Das gleiche Verhalten kann auch anders implementiert werden
- Beispiele (siehe Skript)

Beispiel *break*

```
#include <stdio.h>
int main()
{
    int i;

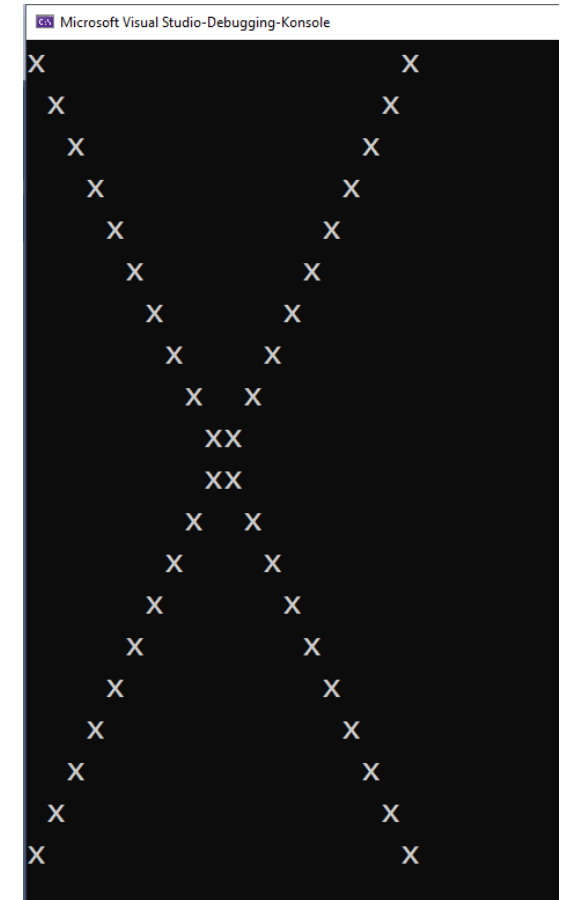
    for (i = 0; i < 10; i++) {
        printf("%d\n", i);
        if (i == 5) break;
    }

    return 0;
}
```

Verschachtelte Schleifen

- Schleifen können auch in einander verschachtelt werden

```
// Kreuz ausgeben
for (i = 0; i < 20; i++) {
    for (j = 0; j < 20; j++) {
        if ((i== j) || (i==19-j)) {
            // gebe 'x' aus falls i und j gleich
            // bzw. i und (19-j) gleich
            printf("x");
        }
        else printf(" "); // Leerzeichen ausgeben
    }
    printf("\n");
}
```



Übung: Lottoschein

- Geben Sie die Zahlen von 1 bis 49 wie auf einem Lottoschein aus!
- Nutzen Sie verschachtelte Schleifen!
- Die Ausgabe soll so aussehen:

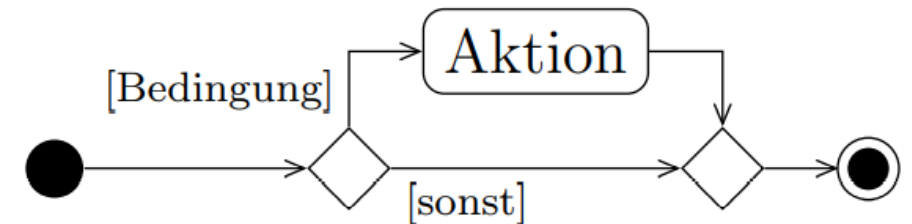
```
Lottoschein:  
 1  2  3  4  5  6  7  
 8  9 10 11 12 13 14  
15 16 17 18 19 20 21  
22 23 24 25 26 27 28  
29 30 31 32 33 34 35  
36 37 38 39 40 41 42  
43 44 45 46 47 48 49
```

Tipp: Schöner wird es mit `"%3d"`



Bedingte Verarbeitung (if-Anweisung)

- Wenn die Bedingung *wahr* ergibt, so wird der Codeblock (Aktion) ausgeführt
- Wenn die Bedingung *nicht wahr* ergibt, wird der Block ausgelassen



if (Bedingung) Aktion

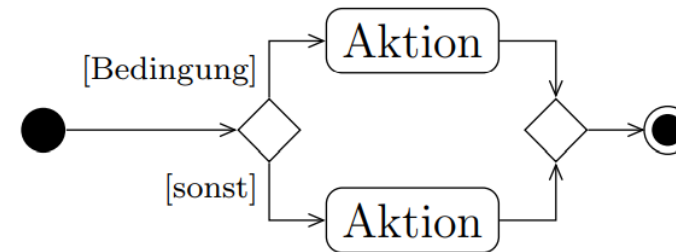
Beispiel:

```
if (i%7==0) printf("\n"); // Zeilenvorschub bei Vielfachem von 7
```

Damit könnte man auch den
Lottoschein ausgeben!

If-else-Verzweigung

***if** (Bedingung) Aktion*
***else** Aktion*



Beispiel:

```
Kontostand = -345.23;  
if (Kontostand >= 0)  
    printf("Im gruenen Bereich\n");  
else  
    printf("Im Dispo\n");
```


Verschachtelte Kontrollstrukturen

- Verzweigungen können wie alle Kontrollstrukturen beliebig verschachtelt werden

```
#include <stdio.h>
int main()
{
    int Monat = 2; // Februar
    int Schaltjahr = 0; // Schaltjahr: 1=ja; 2=nein
    int Tage; // Kalendartage im Monat

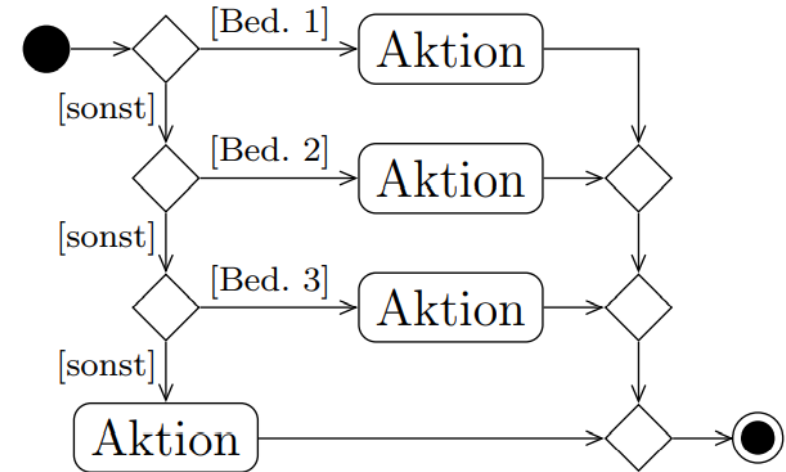
    if (Monat == 2)
        if (Schaltjahr) Tage = 29;
        else Tage = 28;
}
```

```
if (Monat == 2) {
    if (Schaltjahr) Tage = 29;
    else Tage = 28;
}
```

Auf welche if-Anweisung bezieht sich das *else*? → Innere if-Anweisung
Für die Lesbarkeit auf gleiche Tiefe einrücken!
Achtung: Compiler berücksichtigt Einrückung nicht!
Anderer Weg: Blöcke mit {} definieren

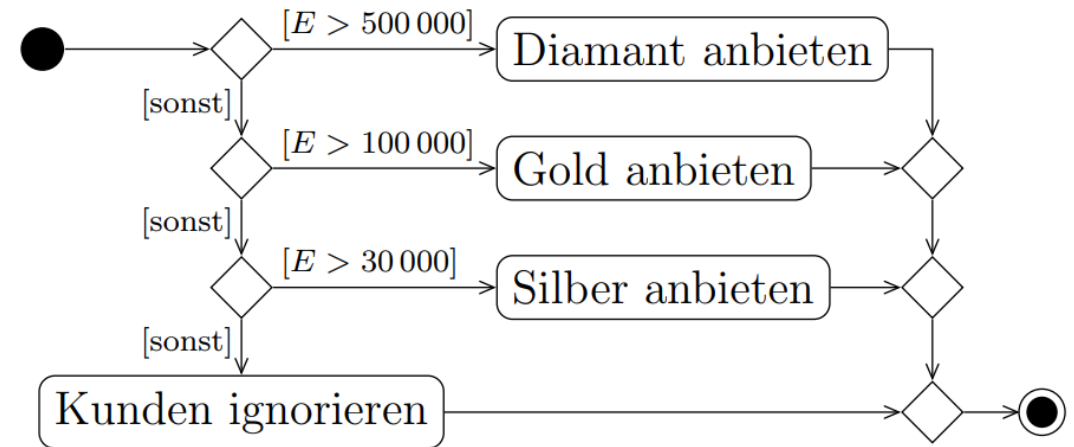
Verkettete Verzweigungen (*else-if*)

- Mehrere if-Befehle können verkettet werden, so dass verschiedene Bedingungen nacheinander geprüft werden



Beispiel: *else-if*

- Juwelier bietet seinem Kunden unterschiedlichen Schmuck abhängig vom Einkommen an

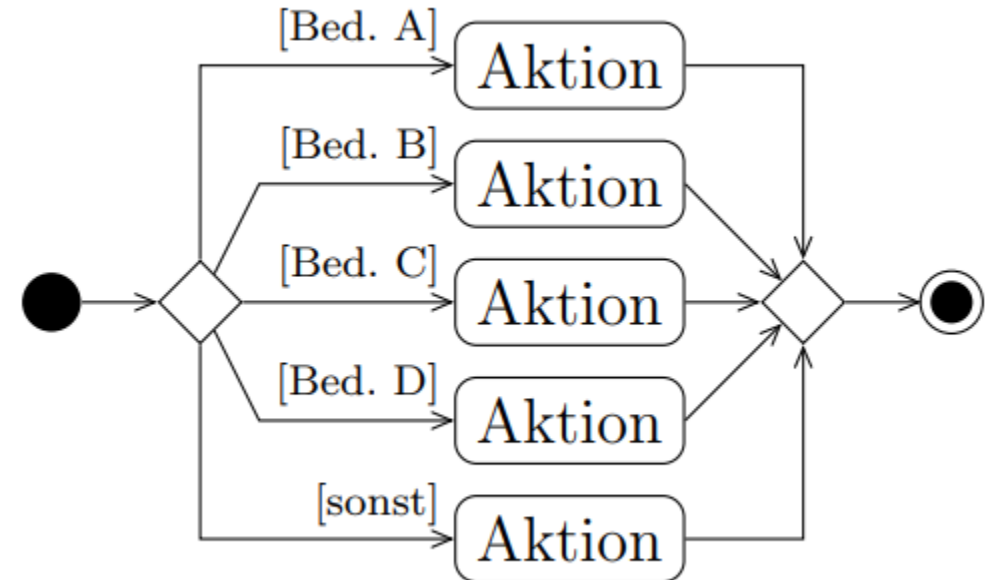


```
if (E > 500000) printf("Darf es ein Diamant für Sie sein?\n");  
else if (E > 100000) printf("Ich habe einen Goldring für Sie.\n");  
else if (E > 30000) printf("Kaufen Sie Silber!\n");  
else printf("Ich habe gerade leider keine Zeit fuer Sie.\n");
```

Mehrfache Alternative (*switch-case*)

- Auswahl zwischen mehr als zwei gleichberechtigten Möglichkeiten

```
switch( Variable ) {  
  case Konstante1: Aktion [break;]  
  [case Konstante2: Aktion [break;]  
  [case Konstante3: Aktion [break;]  
  [...]  
  [default: Aktion [break;]  
}
```



Beispiel: *switch-case*

Ausdruck muss von integralem Typen sein (d.h. Ganzzahl)
Aber auch Zeichen (*char*) sind in C Ganzzahlen

```
// Beispiel switch-case
int Jahreszeit = 2; // Jahreszeiten sind mit 1-4 durchnummeriert
switch (Jahreszeit) {
    case 1: printf("Fruehling\n"); break;
    case 2: printf("Sommer\n"); break;
    case 3: printf("Herbst\n"); break;
    case 4: printf("Winter\n"); break;
    default: printf("Unbekannte Jahreszeit\n"); break;
}
```

Alternativen sind mit *case* angegeben und werden mit *break* beendet

Über das Einrücken lässt sich bei *switch-case* durchaus streiten!

Falls keine Alternative zutreffend ist, wird gesprungen zu *default*

Beispiel 2: *switch-case*

```
switch (Monat) {  
  case 4:           /* April */  
  case 6:           /* Juni */  
  case 9:           /* September */  
  case 11: Tage = 30; break; /* November */  
  case 2:           /* Februar */  
    if (Schaltjahr) Tage = 29;  
    else Tage = 28;  
  break;  
  default: Tage = 31; break; /* sonst */  
}
```

break ist nicht zwingend erforderlich, sondern es werden auch die anderen „cases“ bis zu einem *break* durchlaufen

In diesem Fall bei 4,6 und 9 bis case 11

Achten Sie auch hier auf die Lesbarkeit!

Häufig wird jeder *case* mit einem *break* abgeschlossen um den Code „einfach“ zu halten

Übung (in 2er Teams)

- Was gibt es heute in der Mensa?
 - <https://www.studierendenwerk-hamburg.de/speiseplan-nocache?t=today>
- Bestimmen Sie den Preis für ein Essen abhängig vom gewählten Menü und der Preiskategorie (Studi, Mitarbeiter, Gast)
- Nutzen Sie Integer-Werte zur Unterscheidung (z.B. 0 für Suppe, 1 für Menü 1, ..)
- Nutzen Sie sowohl *if* als auch *switch in ihrer/ihren Lösung(en)*
- Beschränken Sie sich auf nur 3 unterschiedliche Menüs

Bedingter Ausdruck

- In der letzten Vorlesung war auch von ternäre Operatoren kurz die Rede
- In C gibt es als ternären Operator den bedingten Ausdruck, welcher sich wie eine *if-else*-Anweisung verhält

Ausdruck1 ? Ausdruck2 : Ausdruck3

Falls *Ausdruck1* wahr ist, wird *Ausdruck2* ausgeführt ansonsten *Ausdruck3*

Beispiel:

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

Verhält
sich wie

```
if (a > b) z = a;  
else z = b;
```



Ermittlung des Maximalwerts

Absolute Sprünge (bitte nicht mehr nutzen!)

- Mit *goto* können Sie zu einer anderen Stelle im Code innerhalb einer Funktion springen
- Diese Stelle wird mit einer Marke angegeben
- Im Beispiel (links) übergibt eine *goto-Anweisung* die Steuerung an den Punkt mit der Bezeichnung *stop*, wenn $i==5$
- Goto wird noch unterstützt, es gilt aber als „verpönt“
- Vermeiden Sie *goto*-Befehle!

```
// goto.c
#include <stdio.h>

int main()
{
    int i, j;

    for (i = 0; i < 10; i++)
    {
        printf_s("Outer loop executing. i = %d\n", i);
        for (j = 0; j < 3; j++)
        {
            printf_s(" Inner loop executing. j = %d\n", j);
            if (i == 5)
                goto stop;
        }
    }
    /* This message does not print: */
    printf_s("Loop exited. i = %d\n", i);

stop: printf_s("Jumped to stop. i = %d\n", i);
}
```

Quelle: <https://docs.microsoft.com/de-de/cpp/c-language/goto-and-labeled-statements-c?view=msvc-170>

Praktikum 2

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!