[Steuerung] Übersicht Programmablaufsteuerung

- Konditionale Verzweigung: *if - else*
- *switch*-Anweisung
- Schleifenkonstrukte:
  while, do - while
- *for* – Schleife
- Sprung-Anweisungen:
  break, continue, goto, return

[Steuerung] Anweisungen

- Einfachste Form einer Anweisung:

```
Ausdruck ;
```

```
a = b * c ;
j++ ;
```

- Zusammenfassung mehrerer Anweisungen

```
{
    Anweisung1 ;
    Anweisung2 ;
}
```

Der geklammerte Block gilt dann als eine Anweisung.

[Steuerung] Anweisungen

- Es gibt auch leere Anweisungen (die sinnvoll sein können)

```
while (getchar() != '\n')
{
}
```

- In einem Block können als erstes Variablen vereinbart werden (und auch nur da).

- Ein Block (als Anweisung) wird nicht mit einem Semikolon abgeschlossen.

[Steuerung] Bedingte Anweisungen: if - else

- Syntax

```
if ( bedingung )
    anweisung1
[ else
    anweisung2 ]
```

- else ist optional, Anweisung kann ein Block oder auch eine Anweisung sein.

[Steuerung] if – else Beispiel

```
if (n > 0)
    if (n % 2)
        printf(“positiv, ungerade\n“);
else
    printf(“nicht positiv“);
```

[Steuerung] if – else Beispiel

```
if (n > 0)
    if (n % 2)
        printf(“positiv, ungerade\n“);
else
    printf(“nicht positiv“);
```

```
if (n > 0)
    if (n % 2)
        printf(“positiv, ungerade\n“);
    else ;
else
    printf(“nicht positiv“);
```

```
if (n > 0)
{
    if (n % 2)
        printf(“positiv, ungerade\n“);
}
else
    printf(“nicht positiv“);
```

[Steuerung] Bedingte Anweisungen: switch

- Eine Verzweigung

```
switch ( bedingung )
{
    case wert1:
        anweisungsfolge1
    case wert2:
        anweisungsfolge2
    ..
    case wertN:
        anweisungsfolgeN
    [default:
        anweisungsfolgeD ]
}
```

- wert1...wertN sind Konstante oder konstante Ausdrücke
- Die Reihenfolge von wert1...wertN ist beliebig
- Jede case- und default Klausel darf eine Anweisungsfolge beinhalten. Keine Klammern.

[Steuerung] switch - Beispiel

```
int c;
c = getchar();
switch(c)
{
    case 'a':
        printf(“ein kleines a, k“);
    case 'A':
        printf(“ein grosses a\n“);
        break;
    default:
        printf(“kein kleines und kein grosses a“);
}
```

- Beachten sie den Effekt der break-Anweisung!

[Steuerung] Schleifen: while, do - while

```
while (bedingung)
    anweisung;
```

```
do
    anweisung
while (bedingung);
```

```
int c;
do
{
    c = getchar();
    putchar(c);
} while (c !='e');
```

- Sehr ähnlicher Steuerungsmechanismus, eine do-Schleife wird aber mindestens einmal durchlaufen.

[Steuerung] Schleifen: for

```
for ( ausdruck1 ; bedingung ; ausdruck2)
    anweisung;

ausdruck1: initialisierungsausdruck
ausdruck2: Iterierungsausdruck


äquivalent zu:

ausdruck1;
while (bedingung)
{
    anweisung;
    ausdruck2;
}
```

- Initialisierungs- und Iterierungsausdruck dienen meist der Initialisierung und Inkrementierung einer *Laufvariablen*

[Steuerung] Schleifen: for

```
int i;
for ( i = 0 ; i < 5 ; i++ )
{
    printf(“%d\n“,i);
}
```

```
int i , int j;
for (i = 0 , j = 10 ; i < j ; i++ , j-- )
{
    printf(“%d %d\n“,i,j);
}
```

```
/* eine besondere Schleife */
int i;
for (i=0 ; ; i++)
{
    printf(“%d\n“,i);
}
```

[Steuerung] Sprünge: break

- Implizite Sprünge: in Kontrollstrukturen nach testen der Bedingung

- break
  in einer Schleife oder einer switch-Anweisung bewirkt *break* einen Sprung „hinter“ die Schleife

```
for (i = 0 ; i < strlen(mystring) ; i++ )
{
    if (mystring[i] == '\0')
        break;
    mystringcopy[i]=mystring[i];
}
```

[Steuerung] Sprünge: continue

- continue
  Beendet einen aktuellen Schleifendurchlauf. Veranlasst einen Sprung zum Testausdruck
  (bei einer *for* – Schleife wird noch der Iterationsausdruck ausgeführt)

```
char mystringcopy[20];
char mystring[] = “RinderWahn“;

for (i = 0 , j=0 ; i < strlen(mystring) ; i++ )
{
    if (isupper(mystring[i]))
        continue;

    mystringcopy[j++]=mystring[i];
}
```

[Steuerung] Sprünge: goto

- goto
  Expliziter Sprung zu einer Markierung im Code

```
for (i=0 ; i < LIMIT1 ; i++){
    for (j=0 ; j < LIMIT2 ; j++){
        ... if (somethingbadhappend) goto loop_exit;
    }
}
loop_exit:;
```

Solche Sprunganweisungen führen sehr schnell zu unüberschaubarem Code und schwer nachvollziehbarem Verhalten.

goto im allgemeinen vermeiden, im Kurs gänzlich unterlassen!

[Steuerung] Sprünge: return

- return
  Anweisung zum Beenden einer Funktion. Der Programmfluß kehrt zur aufrufenden Funktion zurück.
  Die return-Anweisung kann einen Wert liefern, welcher dann dem Wert der beendeten Funktion entspricht.