Bedingungen:

einfache Form:

```
if (ausdruck)
{
    Anweisungsblock
}
```

Der Ausdruck muß einen arithmetischen Typ oder einen Zeigertyp liefern. Ist das Ergebnis ungleich 0, wird der Anweisungsblock ausgeführt, andernfalls nicht.

allgemeine Form:

```
if (ausdruck)
{
    Anweisungsblock1
}
else
{
    Anweisungsblock2
}
```

Mehrfachauswahl:

```
switch (ausdruck)
{
    case wert1: anweisungen1;
    case wert2: anweisungen2;
    ...
    case default: standardanweisungen;
}
```

Der Ausdruck muß einen Integertyp besitzen. Er kann auch Zuweisungen enthalten (siehe Hinweis). Alle Anweisungen, ab dem zutreffenden Fall werden ausgeführt. Dies wird durch die Anweisung „`break`“ verhindert.

Bedingungsoperator:

```
ergebnis = (ausdruck) ? wert1 : wert2;
```

Verknüpfen von Bedingungen:

UND = &&
ODER = `||`
Es wird von links geprüft. Wenn das Ergebnis feststeht, wird nicht weitergeprüft.

Hinweis:

Der Ausdruck, der die Bedingung einer Selektion darstellt, kann Zuweisungen enthalten.
Bsp: `if (p=positiv(x))`
Hier enthält die Variable p anschließend den Wert, den die Funktion „positiv“ zurückliefert.
Solch eine Programmierung ist aber unübersichtlich, schlecht überschaubar und daher zu vermeiden!

Schleifen:

Prüfen der Abbruchbedingung am Anfang:

```
while (bedingung)
{
    anweisungsblock;
}
```

`bedingung` kann Zuweisungen enthalten. Dies ist aber nur in speziellen Fällen sinnvoll.

Prüfen der Abbruchbedingung am Ende:

```
do
{
    anweisungsblock
}
while (bedingung);
```

`bedingung` kann Zuweisungen enthalten. Dies ist aber nur in speziellen Fällen sinnvoll.

Zählschleife:

```
for (anweisung1; bedingung; anweisung2)
{
    anweisungsblock;
}
```

1. Führe anweisung1 aus
2. Wenn bedingung wahr ist, dann fahre mit 3. fort, ansonsten mit 6.
3. Führe Anweisungsblock aus
4. Führe Anweisung2 aus
5. Fahre mit 2. fort
6. Fahre mit den Anweisungen hinter der Schleife fort

Bsp: `for (x=1; x<10; x++)`

Die Anweisungen können auch mehrere Zuweisungen enthalten, die durch Komma getrennt werden, dies macht aber den Algorithmus meist nur unübersichtlich und ist daher i.allg. zu vermeiden. Anweisung1 kann auch fehlen, falls z.B. der Variablen bereits vorher ein Wert zugewiesen wurde.

continue:

durch die Anweisung „`continue`“ wird sofort zum Schleifenkopf gesprungen, ohne die restlichen Anweisungen innerhalb des Anweisungsblocks auszuführen. Dies macht aber den Algorithmus meistens unübersichtlicher und sollte daher vermieden werden.

break:

Bricht die Schleife ab. Auch diese Anweisung führt im Regelfall dazu, daß der Algorithmus schlechter durchschaubar wird und ist daher normalerweise zu vermeiden.

Datentypen

Bedeutung des Datentyps: Interpretation der Nullen und Einsen

Ganzzahlige Typen:

Datentyp	Größe	Zahlbereich
short	1 Byte	-128 ... 127
unsigned short	1 Byte	0 ... 255
int	2 Byte	-32768 ... 32767
unsigned int	2 Byte	0 ... 65535
long	4 Byte	-2147483648 ... 2147483647
unsigned long	4 Byte	0 ... 4294967295

Boolesche Werte:

In C nicht vorhanden. Stattdessen int (oder anderer ganzzahliger Typ)

Zeichen:

Datentyp	Größe	Zahlbereich
(unsigned) char	1 Byte	0 ... 255
signed char	1 Byte	-128 ... 127

Fließkommazahlen:

Datentyp	Größe	Genauigkeit
float	4 Byte	6 Stellen
double	8 Byte	10 Stellen
long double	10 Byte	10 Stellen

Zahlbereich: jeweils mindestens 10^{-38} ... 10^{38}

Aufzählungstypen:

enum ist Ganzzahltyp mit eingeschränktem Wertebereich. Zum Bsp.
enum Woche{SO,MO,DI,MI,DO,FR,SA};
Woche tag;
tag = MI;

Typendeklaration:

typedef

Typumwandlung:

Mittels Cast-Operator: (typ)variable

Operatoren:

Grundrechenarten:

\+ = Addition
\- = Subtraktion
* = Multiplikation
/ = Division (bei Integertypen werden die Nachkommastellen abgeschnitten)
% = Modulo

Zuweisungsoperatoren:

`+=: x += 2` entspricht `x = x +2`
`-=: x -= 2` entspricht `x = x -2`
`*=: x *= 2` entspricht `x = x *2`
`/=: x /= 2` entspricht `x = x /2`
`%=: x %= 2` entspricht `x = x %2`

Inkrementieren und Dekrementieren:

Präinkrement: `++x` `printf („%u“, ++x);`
Postinkrement: `x++` `printf („%u“, x++);`
Prädekrement: `--x` `printf („%u“, --x);`
Postdekrement: `x--` `printf („%u“, x--);`

Die einzelnen Anweisungen geben für einen Startwert x=10 die Werte 11, 10, 9, bzw. 10 aus und x enthält im Anschluß den Wert 11, 11, 9, bzw. 9.
Diese Operatoren sollten nur für das Hoch- und Runterzählen von Zählern verwendet werden.

Hinweis:

Die Operatoren können verschachtelt werden. Dies kann aber leicht zu unübersichtlichen Code führen.
Bsp: `x=12; x += ++x+x++; x = x+x;` Welchen Wert hat x?