# Operatoren

## Operatoren

Operatoren sind Rechenzeichen. Sie sagen dem Rechner, was für eine Art der Rechnung mit den Operanden ausgeführt werden soll. Je nach Kontext kann das gleiche Zeichen eine andere Bedeutung haben.
Operatoren können miteinander verkettet werden um kompliziertere Ausdrücke zu erstellen.
z.B.: $a * (b + c)$

| Operatorart | Operatoren |
|---|---|
| Arithmetisch | + - * / % |
| Relational | < <= > >= == != |
| Logisch | && \| ! |
| Adresse | & |
| Referenz | * |
| Bitweise | & \| ~ << >> |

## Ausdrücke

Ein Ausdruck beschreibt, wie ein Wert berechnet wird.

- Ausdrücke werden mit ; abgeschlossen zu einer Anweisung (Befehl)
- z.B.: $a * (b + c)$;
  Die einfachsten Ausdrücke sind:
- Variablen: Ein Wert der berechnet wird, wenn das Programm ausgeführt wird.
- Konstanten: Ein Wert, der sich nicht ändert.

## Begriffe

- Präzedenz (Bindungsstärke) beachten
  - Klammer vor Punkt vor Strich
  - Klammern bei Bedarf setzen
- Assoziativität
  - Links-assoziativ: $1 * 2/3$ bedeutet $(1 * 2)/3$

- Auswertungsreihenfolge meistens nicht festgelegt

## Relationale Operatoren

- Vergleichen zwei Operanden
- Ergeben 0 (falsch) oder 1 (wahr)
  - In C gibt es keinen expliziten Typ für Wahrheitswerte (Boolesche Werte)
  - 0 ist falsch und alle anderen Werte sind wahr

## Logische Operatoren

**Logisches UND** &&

- $a == 1$ && $b == 2$ ist wahr, wenn a gleich 1 und b gleich 2 ist.
  **Logisches ODER** ||
- $a == 1$ || $b == 2$ ist wahr, wenn a gleich 1 oder b gleich 2 ist.
  **Logische Negation** !
- $!(a == 0)$ ist wahr wenn a ungleich 0 ist
  - Entspricht: $a! = 0$

## Zuweisungsoperatoren

- $x = y;$
  - Berechne y, kopiere das Ergebnis nach x
  - summe = summe + zahl;
- Zuerst wird das Ergebnis links des Operators = berechnet
  - summe + zahl
- Danach wird das Ergebnis zugewiesen
- L-Wert = R-Wert
  - Schachteln von Zuweisungen möglich
  - Rechts-Assoziativ

```
int i = 5;
j = i = 6;
l = j - (i = 3); // schlechter Stil!
```

### Weitere Zuweisungsoperatoren

$* =$, $/ =$, $- =$, (*Komprimieren Ausdrücke*)
$x+ = 2;$ entspricht $x = x + 2;$

++ und --

- Inkrement-Dekrement-Operator
- $++x$ entspricht $x+=1$
- $x++$ entspricht $x+=1$, liefert aber den Wert von x vor der Erhöhung zurück.

# Komma-Operator

- Mit , können mehrere Anweisungen in einen Ausdruck verpackt werden.

```
int i = 5;
int j = (i = 3, i + 5);
```

- Wird selten gebraucht, am häufigsten in Schleifen

# Selektion

## if-Anweisungen

if-Anweisungen erlauben Sequenzen nur dann auszuführen, wenn eine Bedinungung erfüllt ist.

- Die if-Anweisung startet einen neuen Block { ... }

```
if(Bedingung)
{
    Anweisung1; //wird ausgeführt wenn
    Anweisung2; //Bedingung ungleich 0 ist
}
```

- if-Anweisungen mit einzeiliger Sequenz brauchen keine Klammern

```
if(Bedingung)
    Anweisung1;
```

## if-else-Anweisungen

Wenn die Bedingung nicht erfüllt ist, wird die alternative Sequenz des else-Blocks ausgeführt.

```
int examPts = ...;
int exercisePts = ...;

if(examPts > 35 && exercisePts + examPts > 50)
    printf("Positive Note\n");
else
    printf("Negative Note\n");
```

## Schachteln von Bedingungen

- Bedingungen können beliebig oft ineinander verschachtelt werden

```
if(grade >= 1 && grade <= 5)
{
    printf("Valid grade\n");
    if(grade == 5)
        printf("Failing course\n");
    else
        printf("Passing course\n");
}
else
    printf("Invalid grade\n");
```

## Mehrfache Alternativen

Mit `else if` können auch mehrere Bedingungen nacheinander überprüft werden. Dabei wird nur die **erste Sequenz**, deren **Bedingung erfüllt wurde** ausgeführt.

```
if(grade == 1)
    printf("Sehr gut\n");
else if(grade == 2)
    printf("Gut\n");
else if(grade == 3)
    printf("Befriedigend\n");
else if(grade == 4)
    printf("Genuegend\n");
else if(grade == 5)
    printf("Nicht genuegend\n");
else
    printf("Ungueltige Note\n");
```

## Funktionales if-else

- Auch **ternärer** Operator genannt.
- `Bedingung ? if-Teil : else-Teil`
- Wenn die Bedingung erfüllt ist, wird der if-Teil ausgeführt, ansonsten der else-Teil

```
int abs = i < 0 ? -i : i;
int max = i > j ? i : j;
int min = i < j ? i : j;
```

# Switch-Case-Anweisungen

- Alternative zu mehrfachen Alternativen

```
switch(grade)
{
    case 1: printf("Sehr gut\n");
    break;
    case 2: printf("Gut\n");
    break;
    case 3: printf("Befriedigend\n");
    break;
    case 4: printf("Genuegend\n");
    break;
    case 5: printf("Nicht genuegend\n");
    break;
    default: printf("Ungueltige Note\n");
}
```

## Semantik von switch

- switch ist ein Sprung zum case, die Ausführung wird im entsprechenden case oder default (wenn keiner der Fälle passt) fortgesetzt.
  - case nur für konstante Werte, das heißt keine variablen Fälle oder Vergleiche (case j oder case i < j funktionieren nicht)
  - Solche goto-ähnlichen Anweisungen sind nicht strukturiert
  - case entspricht nicht der if-Anweisung, sondern dem Label eines Sprungs (Spring zu Stelle xyz)
  - Ohne break wird die Ausführung im nächsten case fortgesetzt (fall-through)
- break; unterbricht die Ausführung

### fall-through

- Hat month den Wert 4 wird zuerst der Fall 4 abgehandelt, dann der Fall 8, dann der Fall 12 bevor die Ausführung auf ein break trifft
- Hat month den Wert 12 wird nur der Fall 12 ausgeführt, danach folgt direkt ein break

```
int daysOfMonth = -1;

switch(month)
{
    case 1:
    case 4: daysOfMonth = 30;
    case 8:
    case 12: daysOfMonth = 31;
    break;
    case 6: daysOfMonth = 30;
    break;
    ....
}
```

### Anwendungsfälle

- Gut geeignet für kurze Variablenzuweisungen oder Ausgaben.

- fall-through beachten

```
switch(input)
{
    case 'W': --y;
    break;
    case 'A': --x;
    break;
    case 'S': ++y;
    break;
    case 'D': ++x;
}
```