Float-Arithmetik

- Implementiert IEEE 754-1985 Standard
- Überlauf ("overflow"):
  - Zahl wird zu groß / zu klein
  - Beispiel: max.float * 2
  - Resultat = $+\infty$ bzw. $-\infty$
- Underflow:
  - Zahlen liegen zu dicht an der 0
  - Resultat = +0.0 bzw. -0.0
- NaN (Not a Number):
  - Rechnungen, wo kein sinnvoller Wert rauskommt
  - Bsp.: 1/0 , ∞*0 , sqrt(-1.0)

Beispiel: Quadratische Gleichungen

- Lösen der quadratischen Gleichung $x^2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

```
#include <math.h>
…
sqroot = sqrt(b*b - 4.0*c)
root1 = (-b + sqroot) / 2.0
root2 = (-b - sqroot) / 2.0
```

Vergleichsoperatoren

- Operanden müssen gleichen Typ haben
- Resultat: `bool`
- Achtung: verwechsle nicht `=` und `==` !
  - `if ( a = b )` liefert keine Fehlermeldung des Compilers!
  - Neuere Compiler bringen immerhin eine Warnung, falls richtiges Flag eingeschaltet.

Richtiger Vergleich von Floating-Point-Werten

- Verwendung von == ist fast immer ein Bug!
  - Bsp.: in Qt 3.1 findet man (qwmatrix.h)

```
bool isInvertible() {
    return (m11*m22 - m12*m21) != 0;
}
```

- Meistens wollen wir "Gleichheit bis auf Rundungsfehler"
- Bessere Technik: "*epsilon guard*"

```
if ( fabsf(f1 - f2) < 1E-5 )
{
    // Gleichheit
}
```

- Noch besser:

```
if ( fabsf(f1 - f2) < 1E-5*f1 )
```

Der ternäre Operator (Bedingungsoperator)

- `? :`
- Eigentlich Kontrollflußkonstrukt, verkappt als Operator
- Definition:

cond* ? *true-expr* : *false-expr

- Wert des Ausdrucks =

Wert von ***true-expr*** , falls Wert von ***cond*** ≠ 0
Wert von ***false-expr*** , falls Wert von ***cond*** = 0

- Beispiele:

```
puts( x ? "true" : "false" );
```

```
double max = (a>b) ? a : b;
```

```
printf("Bestellung: %d %s\n",
       i,
       i == 1 ? "Baum" : "Bäume" );
```

- Beachte: nur der benötigte Ausdruck wird ausgewertet
- Beispiel:

```
int i = true ? 1 : 1/0;
```

gibt *nie* eine Division-by-Zero Exception


Präzedenz und Bindung

- Englisch: "precedence" & "associativity" ("grouping")
- Ohne diese ist ein Ausdruck der Art

  `a+b*c`

  nicht eindeutig
- Präzedenz: Reihenfolge der Auswertung der Operatoren
- Assoziativität: Reihenfolge bei Operatoren gleicher Präzedenz


Präzedenz	Operator	Grouping
Höchste	[] () . ->	L-R
	++ -- ~ ! + -	R-L
	(cast)	R-L
	* / %	L-R
	+ -	L-R
	<< >>	L-R
	< … >=	L-R
	== !=	L-R
	&	L-R
	^	L-R
	|	L-R
	&&	L-R
	||	L-R
	?:	L-R
	= += … |=	R-L
Niedrigste	,	L-R

Auswertungsreihenfolge

- Reihenfolge der Auswertung der Operanden nicht festgelegt im Standard!
- Bsp.:

```
i = 2;
j = (i++) * (i--);
j = ?
```

- Resultat:
  - 6 oder 2
- Fazit: Keine Operanden mit Nebeneffekten!
  - Nur dann gilt Kommutativität!
  - Ist sowieso schlechter Stil!

Short-cut logic in Boole'schen Ausdrücken

- Shortcuts bei Boole'schen Ausdrücken
  - || und && werden von links nach rechts ausgewertet
  - Falls Wahrheitswert feststeht, keine weitere Auswertung!
  - `true || x → true`
  - `false && x → false`
  - Im Standard festgelegt
- Kann sehr praktisch sein, kann Performance steigern
- Nicht einsetzen in komplexen Ausdrücken
- Dokumentieren!
- Beispiel:

```
if ( foo(x) && foo(y) )
```

Promotion

- Gemischte Ausdrücke eingebauter Typen
- Automatische Konvertierung zum "höchsten" Typ (*Promotion*)

- Promotion-Hierarchie:

```
long double
double
float
unsigned long int
long int
unsigned int
int
unsigned short int
short int
unsigned char
char
bool
```


- Promotion findet von innen nach außen statt!
- Beispiel:

```
float f = 1/2;     // f == 0.0 !
```

Anweisungen (Statements)

- Die Bausteine eines Programms mit Effekt (normalerweise)
- Programm ist lange Liste von Anweisungen, getrennt durch Semikolon
- Stil: eine Anweisung pro Zeile (trotzdem Semikolon)
- Anweisung =
  - Leere Anweisung (`;` `;` oder `{` `}`)
  - Ausdruck (nicht alle haben einen Effekt)
  - Deklarationen (Variable, Typen, Funktionen, Klassen, …)
  - Kontrollfluß-Statement (später)

Beispiele

```
int i, j;    // Deklaration ist auch Anweisung
i = j;
i ++ ;       // Nebeneffekt hier die Hauptsache
;            // leeres Statement
x + y;       // sinnlos, aber ok
z = x + y;
```