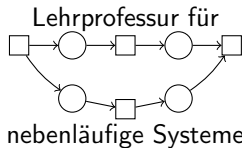


Vorkurs Informatik

Dezimalzahlen / Wiederholungen

Robert Lorenz

Tag 4



Definition double

- Der Datentyp double repräsentiert eine **endliche Teilmenge** aller Dezimalzahlen zwischen $-1.7976931348623158 \cdot 10^{308}$ und $+1.7976931348623158 \cdot 10^{308}$ (jeweils einschließlich).
- Die Zahlen dieser Teilmenge heißen **exakt darstellbar**
- Der Abstand zwischen exakt darstellbaren Zahlen ist (relativ) klein
- Nicht exakt darstellbare Zahlen innerhalb der Wertegrenzen werden **gerundet** (zur nächstgelegenen exakt darstellbaren Zahl)

Das Überschreiten der Wertegrenzen in Rechnungen mit dem Datentyp double führt zu unerwarteten Ergebnissen!

Festkomma-Schreibweise von Dezimalzahl-Konstanten

`<Vorzeichen><Vorkommastellen>.<Nachkommastellen>`

wobei:

- Das Vorzeichen ist + oder - und ist optional.
- Die Vorkommastellen sind eine Ziffernfolge wie zur Darstellung einer ganzen Zahl
- Die Nachkommastellen sind eine Ziffernfolge
- Vorkommastellen oder Nachkommastellen dürfen fehlen

Beispiele: 2.01, -.5, +31., 0.0003

Fließkomma-Schreibweise von Dezimalzahl-Konstanten

$\langle \text{Mantisse} \rangle e \langle \text{Exponent} \rangle$ (oder $\langle \text{Mantisse} \rangle E \langle \text{Exponent} \rangle$)

wobei:

- Die Mantisse wird in Festkomma-Schreibweise oder als Ganzzahl-Konstante angegeben
- Der Exponent wird als Ganzzahl-Konstante angegeben

Beispiele: 2.01e0, -.5E2, +31.e-3, 0.0003E+5

Eine Konstante in der Schreibweise $\langle \text{Mantisse} \rangle e \langle \text{Exponent} \rangle$ hat den Wert $\langle \text{Mantisse} \rangle * 10^{\langle \text{Exponent} \rangle}$

- 5.6e-1 entspricht 0.56
- 0.01e3 entspricht 10

Beispiel

Die Standardbibliothek `float.h` enthält einige symbolische Dezimalzahl-Konstanten:

- `DBL_MAX`: Maximaler Wert für `double`
- `DBL_MIN`: Minimaler Wert für `double`

Ausdrücke vom Typ double

Jede Dezimalzahl-Konstante und jede Variable vom Typ double ist ein **Ausdruck vom Typ double**.

Arithmetische Rechenoperationen

Für Dezimalzahlen stehen dieselben arithmetischen Rechenoperatoren +, -, *, / und % wie für int zur Verfügung. Ausdrücke vom Typ double lassen sich mit diesen Operatoren zur größeren Ausdrücken vom Typ double kombinieren.

Vergleichsoperationen

Für Dezimalzahlen stehen dieselben Vergleichsoperatoren <, <=, >, >=, == und != wie für int zur Verfügung. Ausdrücke vom Typ double lassen sich mit diesen Operatoren zu Bedingungen kombinieren und vergleichen.

Ausdrücke vom Typ double

Ausgabe mit printf

Umwandlungsangaben für double

- %f ist eine Umwandlungsangabe für Dezimalzahlen für eine Ausgabe in Festkommenschreibweise
- %e ist eine Umwandlungsangabe für Dezimalzahlen für eine Ausgabe in Fließkommenschreibweise
- Für jede solche Umwandlungsangabe in <Zeichenkette> muss bei Aufruf eine Dezimalzahl als zusätzlicher Eingabeparameter übergeben werden.

Beispiel

Der Funktionsaufruf

```
printf("Fließkommenschreibweise: %e", 100.5)
```

bewirkt die Ausgabe:

Fließkommenschreibweise: 1.005e2

Typumwandlungen in Wertzuweisungen

Haben Variable und Ausdruck in einer Wertzuweisung

`<Variable> = <Ausdruck>;`

unterschiedlichen Datentyp, so wird der Wert des Ausdrucks (automatisch) in den Datentyp der Variable umgewandelt:

- `int x = 5.5;`
x erhält den Wert 5 (Nachkommastellen werden abgeschnitten!)
- `double x = 5;`
x erhält den Wert 5.0 (Nachkommastellen werden ergänzt, ohne den Wert zu verändern)

Typumwandlungen in Ausdrücken

Kommt in einem Ausdruck ein `double`-Wert vor, so werden alle anderen Werte auch in `double` umgewandelt

Beispiele

- $5 / 2$ hat den Wert 2
- $5.0 / 2$ hat den Wert 2.5

Explizite Typumwandlung

Durch eine sog. **Cast-Operation** können Typumwandlungen erzwungen werden. Ist A ein Ausdruck (irgendeines Typs), so ist $(T) A$ ein Ausdruck vom Typ T.

Beispiele

- $((\text{double}) 5) / 2$ hat den Wert 2.5
- $(\text{double}) (5 / 2)$ hat den Wert 2.0

Die Bibliothek `math.h` enthält einige nützliche vordefinierte und -installierte mathematische Funktionen.

- Der Aufruf von `sin(x)` gibt den Sinus einer Dezimalzahl x zurück.
- Der Aufruf von `cos(x)` gibt den Cosinus einer Dezimalzahl x zurück.
- Der Aufruf von `tan(x)` gibt den Tangens einer Dezimalzahl x zurück.
- Der Aufruf von `sqrt(x)` gibt die Quadratwurzel einer Dezimalzahl x zurück.
- Der Aufruf von `log(x)` gibt den (natürlichen) Logarithmus einer Dezimalzahl x zurück.
- Der Aufruf von `exp(x)` gibt den Wert e^x einer Dezimalzahl x zurück.

Wiederholte Anweisungen

while-Anweisung

Definition

```
1 <A> /* Vorherige Anweisung */  
2 while (<Schleifenbedingung>) { /* Beginn des while-Blocks */  
3   <Wiederholte_Anweisungen>  
4 } /* Ende des while-Blocks */  
5 <B> /* Nachfolgende Anweisung */
```

- Nach <A> wird die <Schleifenbedingung> überprüft
- Ist <Schleifenbedingung> **wahr**, so werden <Wiederholte_Anweisungen> ausgeführt und danach <Schleifenbedingung> erneut überprüft
- <Wiederholte_Anweisungen> werden immer wieder ausgeführt, solange <Schleifenbedingung> bei der Überprüfung als **wahr** ausgewertet wird
- Ist <Schleifenbedingung> **nicht wahr**, so **terminiert** die Schleife und es wird mit fortgesetzt

Wiederholte Anweisungen

while-Anweisung

Beispiel

```
1 int main() {  
2     double x = ((double) rand()) / RAND_MAX * 1e9;  
3     double floor = 0;  
4     while (floor + 1 <= x) {  
5         ++floor;  
6     }  
7     printf("floor(%f) = %f", x, floor);  
8     return 0;  
9 }
```

- Solange $\text{floor} + 1$ kleiner oder gleich x ist (Zeile 4), wird floor um 1 erhöht (Zeile 5)
- Nach Terminierung der Schleife ist floor der größte ganzzahlige Wert, der kleiner x ist
- ++floor entspricht $\text{floor} = \text{floor} + 1$ (Inkrementoperator)

Wiederholte Anweisungen

while-Anweisung

Variablenwerte verfolgen für jeden Schleifendurchlauf:

Beispiel

```
1 double x = 2.3;  
2 double f = 0;  
3 while (f + 1 <= x) {  
4     ++f;  
5 }
```

0.

1.

2.

3.

f == 0

1 <= 2?

f == 1

2 <= 2?

f == 2

3 <= 2?

Wiederholte Anweisungen

Endlosschleifen

Definition

Die Schleifenbedingung muss nach **endlich vielen** Schleifendurchläufen falsch werden, sonst spricht man von einer **Endlosschleife**:

Wert der benutzten Variablen geeignet im `while`-Block ändern!

Beispiel

```
1 int x = 1, y = 0;
2 while (x) { /*wird nie falsch*/
3     y = (y * 11) % 7; /*da x nie veraendert wird*/
4 }
```

Wiederholte Anweisungen

Endlosschleifen

Beispiel

```
1 int x = 0;
2 while (x >= 0) { /*wird nie falsch*/
3     x = (x * 11) % 7; /*da x immer nicht-negativ bleibt*/
4 }
```

Was tun, wenn es doch mal passiert?

Jedes Kommandozeilenprogramm hat eine **Tastenkombination** für den Abbruch von Programmen.

Beispiel Mac OS: [ctrl] + [C]

Wiederholte Anweisungen

for-Anweisung

Definition

```
1 <A> /* Vorherige Anweisung */
2 for (<X>; <Y>; <Z>) { /* Beginn for-Block */
3   <Wiederholte_Anweisungen>
4 } /* Ende for-Block */
5 <B> /* Nachfolgende Anweisung */
```

Diese kompakte Darstellung einer Schleife entspricht:

```
1 <A>
2 <X>;
3 while (<Y>) {
4   <Wiederholte_Anweisungen>
5   <Z>;
6 }
7 <B>
```

Wiederholte Anweisungen

for-Anweisung

for-Schleifen werden üblicherweise verwendet, wenn die Anzahl der Schleifen-Wiederholungen von vornherein feststeht

n-malige Ausführung von Anweisungen

```
1 /* i heisst Zaehlvariable */
2 for (int i = 0; i < 10; ++i) { /* 10 Schleifendurchlaeufe */
3     <Anweisungen>
4 }
```

entspricht

```
1 int i = 0;
2 while (i < 10) {
3     <Anweisungen>
4     ++i; /* i zaehlt die Anzahl der Schleifendurchlaeufe */
5 }
```

Wiederholte Anweisungen

for-Anweisung

Berechnung der Summe der ganzen Zahlen von 1 bis n

```
1 int n = 99;  
2 double sum = 0;  
3 for (int i = 1; i <= n; ++i) {  
4     sum = sum + i;  
5 }
```

- Beginnend mit dem Wert 0 (Zeile 2) wird der aktuelle Wert von `sum` im i -ten Schleifendurchlauf mit i addiert (Zeile 4)
- Nach dem i -ten Schleifendurchlauf hat `sum` den Wert $1 + 2 + \dots + i$

Wiederholte Anweisungen

for-Anweisung

Berechnung der Summe der ganzen Zahlen von 1 bis n

Variablenwerte verfolgen für jeden Schleifendurchlauf

```
1 int n = 2;  
2 double sum = 0;  
3 for (int i = 1; i <= n; ++i) {  
4     sum = sum + i;  
5 }
```

0.
sum == 0
i == 1

1.

1 <= 2?
sum == 1
i == 2

2.

2 <= 2?
sum == 3
i == 3

3.

3 <= 2?

Innere und äußere Schleifen

```
1 <A>
2 while (<Schleifenbedingung_Aussen>) { /* Aeussere Schleife */
3     <Wiederholte_Anweisungen_Aussen_1>
4     while (<Schleifenbedingung_Innen>) { /* Innere Schleife */
5         <Wiederholte_Anweisungen_Innen>
6     }
7     <Wiederholte_Anweisungen_Aussen_2>
8 }
9 <B>
```

- Für **jeden** Durchlauf der **äußeren Schleife** wird die **innere Schleife** mehrmals durchlaufen bis sie terminiert
- Gilt entsprechend für for-Schleifen

Ausgabe eines Quadrats

```
1 for (int i = 1; i <= 5; ++i) {  
2     for (int j = 1; j <= 5; ++j) {  
3         putchar('X');  
4     }  
5     putchar('\n');  
6 }
```

- Der Aufruf `putchar(c)` (siehe `stdio.h`) gibt das Zeichen `c` aus
- Zeilen 2-4: Ein Durchlauf der **inneren Schleife** gibt eine Zeile aus 5 X-Zeichen aus
- Zeile 5: Danach wird in eine neue Zeile gewechselt
- Zeilen 1-6: Im *i*-ten Durchlauf der **äußeren Schleife** wird die *i*-te Zeile mit nachfolgendem Zeilenwechsel ausgegeben (dabei wird die **äußere Schleife** 5-mal durchlaufen)

Wiederholte Anweisungen

Verschachtelte Wiederholungen

Ausgabe eines Quadrats

```
1 for (int i = 1; i <= 5; ++i) {  
2     for (int j = 1; j <= 5; ++j) {  
3         putchar('X');  
4     }  
5     putchar('\n');  
6 }
```

- Zwischenstand der Ausgabe für $i == 1$, $j == 4$:
XXX
- Zwischenstand der Ausgabe für $i == 2$, $j == 1$:
XXXXX
- Zwischenstand der Ausgabe für $i == 3$, $j == 4$:
XXXXX
XXXXX
XXX

Anweisungsblock

Ein Anweisungsblock in C ist von der Form

```
{  
    <Anweisungen>  
}
```

Beispiele:

- Funktionsrümpfe
- if-, else- und else if-Blöcke
- for-, while-Blöcke (NEU!)

Gültigkeitsbereich

Jeder Anweisungsblock erzeugt einen sog. **Gültigkeitsbereich für Variablen**

Definition

- In einem Gültigkeitsbereich deklarierte Variablen heißen für diesen Bereich **lokal**
- Lokale Variablen existieren **nur während der Ausführung** des Anweisungsblocks, also:
 - Reservieren von Speicherplatz durch Deklaration
 - Freigeben von Speicherplatz am Ende des Blocks

Sie können also nur in ihrem Block verwendet werden

Fehlerhafte Benutzung einer lokalen Variable

```
1 {  
2     int x;  
3 }  
4 x = 0;
```

Konventionen für Anweisungsblöcke

- `for`- und `while`-Blöcke beginnen in der ersten Zeile (nicht in der Zeile darunter wie Funktionsrümpfe); Abschließendes `}`-Zeichen in separate Zeile setzen
- Leerzeichen einfügen vor und nach Bedingung eines `while`-Blocks
- `for`-Block: Leerzeichen vor `(` und nach `)` und `;`

In diesem Kapitel wurden u.a. folgende reservierte Schlüsselwörter besprochen:

- `while`, `for`: Schleifen
- `double`: Datentyp für Dezimalzahlen
- `sin`, `cos`, `sqrt`, `log`, `exp`, `putchar`, `tan`:
Bibliotheksfunktionen

ACHTUNG: In C-Programmen wird Klein- und Großschreibung unterschieden!