Matthias Jauernig
Kurs: 03In1

08.12.03

# 9. Aufgabenserie zu den Grundlagen der Informatik

Abgabetermin: Mi, 10.12.03

## Zu 25.) interne Darstellung im *ANSI/IEEE-Format*

### (a) 2.71828 im 4-Byte-ANSI/IEEE-754-Format

- zunächst die benötigten Nachkommastellen (23-Bit-Mantisse) in Dual umrechnen:

| | | | |
|---|---|---|---|
| 0.71828 * 2 | = | 0.43656 | 1 |
| 0.43656 * 2 | = | 0.87312 | 0 |
| 0.87312 * 2 | = | 0.74624 | 1 |
| 0.74624 * 2 | = | 0.49248 | 1 |
| 0.49248 * 2 | = | 0.98496 | 0 |
| 0.98496 * 2 | = | 0.96992 | 1 |
| 0.96992 * 2 | = | 0.93984 | 1 |
| 0.93984 * 2 | = | 0.87968 | 1 |
| 0.87968 * 2 | = | 0.75936 | 1 |
| 0.75936 * 2 | = | 0.51872 | 1 |
| 0.51872 * 2 | = | 0.03744 | 1 |
| 0.03744 * 2 | = | 0.07488 | 0 |
| 0.07488 * 2 | = | 0.14976 | 0 |
| 0.14976 * 2 | = | 0.29952 | 0 |
| 0.29952 * 2 | = | 0.59904 | 0 |
| 0.59904 * 2 | = | 0.19808 | 1 |
| 0.19808 * 2 | = | 0.39616 | 0 |
| 0.39616 * 2 | = | 0.79232 | 0 |
| 0.79232 * 2 | = | 0.58464 | 1 |
| 0.58464 * 2 | = | 0.16928 | 1 |
| 0.16928 * 2 | = | 0.33856 | 0 |
| 0.33856 * 2 | = | 0.67712 | 0 |
| | … | | |

⇒ daraus ergibt sich die folgende Schreibweise (gerundet auf 24 Dualstellen):
$2.71828_{10} = 10.10110111111000010011100_2$

- Normalisierung ergibt:
$10.10110111111000010011100_2 * 2^0 = 1.01011011111100001001100_2 * 2^1$
- Für den Exponenten ergibt sich daher nach Bias-Addition von 127 für eine 4-Byte-Gleitpunktzahl: $1+127 = 128_{10} = 10000000_2$
- Vorzeichenbit: 0 (positive Zahl)
- Daher ergibt sich letztendlich im 4-Byte-ANSI/IEEE-754-Format:

**$2.71828_{10}$ =**

| **0** | **1000 0000** | **010 1101 1111 1000 0100 1100** |
|---|---|---|

**(b) Dezimaldarstellung der folgenden 4-Byte-ANSI/IEEE-754-Zahl:**

| | | |
|---|---|---|
| 1 | 1010 0100 | 101 1101 0010 0000 1100 1000 |

- Vorzeichen: - (da 1 ⇒ negativ)
- Exponent: $10100100_2 = 2^7 + 2^5 + 2^2 = 164_{10}$;
  Abziehen des Bias 127 für 4-Byte-Gleitpunktzahlen: 164-127 = 37
- Mantisse denormalisieren und umrechnen:
  1.101 1101 0010 0000 1100 1000 * $2^{37}$
  = 1101 1101 0010 0000 1100 1000 0000 0000 0000 00 * $2^0$
  = $2^{37} + 2^{36} + 2^{34} + 2^{33} + 2^{32} + 2^{30} + 2^{27} + 2^{21} + 2^{20} + 2^{17}$
  = 2.37434438e11
- Umgerechnete dezimale Zahl: -2.37434438e11

| | | | |
|---|---|---|---|
| **1** | **1010 0100** | **101 1101 0010 0000 1100 1000** | **= -237434437632** |

**Zu 26.)** ***Gleitpunktarithmetik***

Gegeben ist die Lösungsformel einer quadratischen Gleichung der Form $x^2+px+q=0$:

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

Werden $x_1$ und $x_2$ mit dem Computer berechnet, so kann es für p >> q allerdings zu Fehlern kommen, wie die folgenden Beispiele zeigen:

(1) Sei $p=10^8$ und q=10.
Bei der Berechnung mit dem Computer liefert q keinen Anteil, sodass hier als Ergebnis Folgendes herauskommt:

$$x_{1,2} = -\frac{10^8}{2} \pm \sqrt{\frac{10^{16}}{4} - 10} = -5*10^7 \pm \sqrt{2.5*10^{15}} = -5*10^7 \pm 5*10^7$$

$$x_1 = -5*10^7 + 5*10^7 = 0$$

$$x_2 = -5*10^7 - 5*10^7 = -10^8$$

Exakte mathematische Rechnung würde hingegen erbringen:

$$x_{1,2} = -\frac{10^8}{2} \pm \sqrt{\frac{10^{16}}{4} - 10} = -5*10^7 \pm \sqrt{2.5*10^{15} - 10}$$

⇒ Da q=10 vernachlässigt wird, stimmen die vom Computer berechneten Ergebnisse nicht mit den exakten Werten überein!

(2) Sei $p=10^8$ und $q=10+2.5*10^{15}$.
Berechnung mit dem Computer führt aufgrund der „Vernachlässigung“ von q zu folgenden Ergebnissen:

$$x_{1,2} = -\frac{10^8}{2} \pm \sqrt{\frac{10^{16}}{4} - 10 - 2.5*10^{15}}$$

$$= -5*10^7 \pm \sqrt{2.5*10^{15} - 2.5*10^{15}} = -5*10^7 \pm 0$$

$$x_1 = x_2 = -5*10^7$$

Exakte Rechnung würde hingegen ergeben:

$$x_{1,2} = -\frac{10^8}{2} \pm \sqrt{\frac{10^{16}}{4} - 10 - 2.5*10^{15}}$$

$$= -5*10^7 \pm \sqrt{2.5*10^{15} - 2.5*10^{15} - 10} = -5*10^7 \pm \sqrt{-10}$$

Hier hätten $x_1$ und $x_2$ nur folgende komplexe Lösungen:

$$x_1 = -5 * 10^7 + i\sqrt{10}$$

$$x_2 = -5 * 10^7 - i\sqrt{10}$$

sie sind in $\mathbb{R}$ allerdings unlösbar!

⇒ Bei diesen Werten für p und q wiegt der Fehler natürlich besonders schwer, da $x_1$ und $x_2$ exakt mathematisch berechnet nicht reell sind, der Computer dies aber berechnet, was in der Praxis zu schwerwiegenden technischen Fehlern führen kann.

**<u>Hilfe naht:</u> der VIETAsche Wurzelsatz**

Dieser lautet: habe eine quadratische Gleichung $x^2+px+q=0$ die beiden reellen Lösungen $x_1$ und $x_2$, so gilt:

$$x_1 + x_2 = -p$$

$$x_1 * x_2 = q$$

Auf das Problem der verfälschten Werte angewendet ist der Vietasche Wurzelsatz keine eierlegende Wollmilchsau, das ist klar! Sind beide vom Computer mit der „normalen" Lösungsformel berechneten Werte falsch, so kann einem auch Vieta nicht weiterhelfen.
Ist allerdings einer der Werte stark verfälscht, so kann man ihn anstatt mit der Lösungsformel mit dem ersten nahezu korrekt berechneten Wert ausrechnen:

(1) Ist $x_1$ verfälscht und $x_2$ nahezu korrekt berechnet, so ergibt sich $x_1$ aus einer von den beiden folgenden Formeln, die sich aus dem Vietaschen Wurzelsatz ergeben:

$$x_1 = \frac{q}{x_2}$$

$$x_1 = -p - x_2$$

(2) Wenn $x_2$ verfälscht ist und $x_1$ nahezu korrekt berechnet, so ergibt sich $x_2$ aus einer von den beiden folgenden Formeln, die sich wiederum aus dem Vietaschen Wurzelsatz ergeben:

$$x_2 = \frac{q}{x_1}$$

$$x_2 = -p - x_1$$

⇒ Dies ist der Weg, wie man mithilfe des Vietaschen Wurzelsatzes bei Rechnung mit dem Computer zu den beiden mathematisch nahezu korrekt berechneten Lösungen kommt, ohne dass einer oder beide Werte verfälscht erscheinen.

**Zu 27.)** ***NEWTON-Algorithmus***

→ C-Programm, welches mithilfe des NEWTON-Algorithmus einen Näherungswert für die Wurzel einer reellen Zahl berechnet und ausgibt:

```
/*      newton.c -- Matthias Jauernig, 04.12.03                                 */
/*      Programm nutzt das Newtonsche Näherungsverfahren zum Berechnen          */
/*      der Wurzel einer Zahl a                                                 */

#include <stdio.h>

double absol(double zahl){
        if(zahl<0.0)
                return -zahl;
        return zahl;
}


int main(void){
        double x, xneu, eps, rad;
        int nk, i=0;
        //############################ Eingabe ###################################
        printf( "\n-------------------------------------------------------------\n"
                "| Newtonsches Näherungsverfahren zur Berechnung einer Wurzel |\n"
                "-------------------------------------------------------------\n\n");
        do{
                printf("Geben Sie den Radikanden der Wurzel ein (>0): ");
                scanf("%lf", &rad);
        }while(rad<=0.0 && printf("Der Radikand muss größer als 0 sein!\n\n"));
        do{
                printf("Größter Abstand zweier Näherungen (0..%g)?: ", rad/2);
                scanf("%lf", &eps);
        }while((eps<=0.0 && printf("Abstand muss größer 0 sein!\n\n")) ||
            (eps>=rad/2 && printf("Abstand muss kleiner als Radikand/2 sein!\n\n")));
        do{
                printf("Auf wieviele Nachkommastellen soll gerundet werden?: ");
                scanf("%d", &nk);
        }while(nk<1 && printf("Stellenanzahl muss größer als 0 sein!\n\n"));

        //############################ Berechnung ################################
        xneu=rad/2;
        do{
                i++;
                x=xneu;
                xneu = 0.5*(x+(rad/x));
        }while(absol(x-xneu) >= eps);

        //############################ Ausgabe ###################################
        printf( "\n=> Näherungswert für Wurzel(%g) "
                "nach %d Iterationen:  %.*g\n\n", rad, i, nk+1, xneu);

        return 0;
}
```