Matthias Jauernig
Kurs: 03In1

12.12.03
linux-related.de

10. Aufgabenserie zu den Grundlagen der Informatik

oder auch: „Der Programmier-Beleg“ :o)

Abgabetermin: Mi, 17.12.03

Zu 28.) Das k-Dame-Problem

```
/*      schachbrett.c -- Matthias Jauernig, 10.12.03                                */
/*      Programm simuliert ein Schachbrett von n*n Feldern und setzt zufällig
/*      k Damen darauf; dann wird errechnet, welche Damen sich gegenseitig bedrohen */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* -- bedrohen() -- gibt aus, von welchen Damen die auf a[i][j] bedroht wird ------ */
void bedrohen(short **a, const int i, const int j, const int n){
        int k, l;
        short h;
        // zur Vermeidung von doppelten Einträgen werden nur die Damen betrachtet, die
        // weiter rechts bzw. weiter unten stehen, da die Matrix a[][] zeilenweise von
        // links nach rechts abgearbeitet wird; ebenso bedrohen sich nur Damen, zwisch.
        // denen sich keine andere Dame befindet, also muss nach der ersten gefundenen
        // Dame in jeder Richtung Schluss sein
        //###################### 1.) Wird Dame horizontal bedroht? ####################
        for(k=j+1, h=0; k<n && h==0; k++)
                if(a[i][k]==1){
                        printf("[%d][%d]  und  [%d][%d]\n", i, j, i, k);
                        h=1;
                }

        //###################### 2.) Wird Dame vertikal bedroht? ######################
        for(k=i+1, h=0; k<n && h==0; k++)
                if(a[k][j]==1){
                        printf("[%d][%d]  und  [%d][%d]\n", i, j, k, j);
                        h=1;
                }

        //############## 3.) Wird Dame auf der Hauptdiagonalen bedroht? ###############
        for(k=i+1, l=j+1, h=0; k<n && l<n && h==0; k++, l++)
                if(a[k][l]==1){
                        printf("[%d][%d]  und  [%d][%d]\n", i, j, k, l);
                        h=1;
                }

        //############## 4.) Wird Dame auf der Nebendiagonalen bedroht? ##############
        for(k=i+1, l=j-1, h=0; k<n && l>=0 && h==0; k++, l--)
                if(a[k][l]==1){
                        printf("[%d][%d]  und  [%d][%d]\n", i, j, k, l);
                        h=1;
                }
}


/* -- printline() -- gibt eine Linie für die Ausgabe des Schachbretts aus --------- */
void printline(int fl, int n){
        int i, j;

        printf("   ");
        for(i=1; i<=n; i++){
```

```
                printf("+");
                for(j=1; j<=fl; j++)
                        printf("-");
        }
        printf("+\n");
}

/* -- ausg_brett() -- gibt das gesamte Brett aus ------------------------------- */
void ausg_brett(short **a, int n){
        int i, j;
        printf("\n    ");
        for(i=0; i<n; i++)
                printf("%2d  ", i);
        printf("\n");
        for(i=0; i<n; i++){
                printline(3, n);
                printf("%2d ", i);
                for(j=0; j<n; j++)
                        printf("| %s ", (a[i][j]==1) ? "x" : " ");
                printf("|\n");
        }
        printline(3, n);
}


/* -- ANSI C89/C99/C++ verbieten main() ohne Rückgabewert, also "int main(void)"!!! */
int main(void){
        short **a;
        int i, j, k, n, x, y;
        // Zufallsgenerator initialisieren
        srand(time(NULL));

        //######################## Eingabe und Speicherallokierung ####################
        printf( "Bedrohliche Damen\n"
                "-----------------\n\n");
        do{
                printf("Kantenlänge des Schachbretts (Felder)?: ");
                scanf("%d", &n);
        }while(n<1 && printf("=> n muss größer als 0 sein!\n\n"));

        // zunächst Speicher für die Zeiger auf die short-Vektoren allokieren
        a=(short **)malloc(n*sizeof(short *));
        // nun Speicher für jeden short-Vektor allokieren
        for(i=0; i<n; i++){
                a[i]=(short *)malloc(n*sizeof(short));
                // Werte im aktuellen short-Vektor gleich 0 setzen
                for(j=0; j<n; j++)
                        a[i][j]=0;
        }

        do{
                printf("Wieviele Damen sollen sich darauf befinden?: ");
                scanf("%d", &k);
        }while((k<1 && printf("=> k muss größer als 0 sein!\n\n"))
                || (k>n*n && printf("=> Höchstens %d Damen!\n\n", n*n)));

        //############################ Damen aufstellen ###############################
        printf("\n=> Damen befinden sich nun auf folgenden Feldern:\n");
        for(i=1; i<=k; i++){
                do{
                        // x- und y-Koordinate des Feldes zufällig wählen
                        x=rand()%n;
                        y=rand()%n;
                }while(a[x][y]==1); // wenn schon besetzt, dann erneut wählen
                a[x][y]=1;
                if(i%8==0) printf("\n");
                printf("[%d][%d], ", x, y);
```

```
        }
        printf("\b\b \n\n");
        if(n<19)
                ausg_brett(a, n);

        //####################### sich bedrohende Damen ausgeben ######################
        printf("\n=> Felder sich bedrohender Damen:\n");
        // das Array zeilenweise (i) von links nach rechts (j) durchgehen
        for(i=0; i<n; i++)
                for(j=0; j<n; j++)
                        if(a[i][j]==1) // wenn Dame auf Feld, dann schauen, ob bedroht
                                bedrohen(a, i, j, n);
        printf("\n");

        //########################## Speicherfreigabe und Ende ##########################
        // Speicher freigeben: erst einzelne short-Vektoren, dann die Zeiger darauf
        for(i=0; i<n; i++)
                free(a[i]);
        free(a);
        return 0;
}

/* Beispielausgaben:
#########################################################################
Bedrohliche Damen
-----------------

Kantenlänge des Schachbretts (Felder)?: 16
Wieviele Damen sollen sich darauf befinden?: 5

=> Damen befinden sich nun auf folgenden Feldern:
[7][8], [7][15], [8][4], [10][0], [14][8]

     0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 7 |   |   |   |   |   |   |   |   | x |   |   |   |   |   |   | x |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 8 |   |   |   |   | x |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 9 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
10 | x |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
11 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
12 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
13 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
14 |   |   |   |   |   |   |   |   | x |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
15 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

```
=> Felder sich bedrohender Damen:
[7][8]  und  [7][15]
[7][8]  und  [14][8]
[7][15]  und  [14][8]

#####################################################################
Bedrohliche Damen
-----------------

Kantenlänge des Schachbretts (Felder)?: 2
Wieviele Damen sollen sich darauf befinden?: 4

=> Damen befinden sich nun auf folgenden Feldern:
[0][1], [1][1], [0][0], [1][0]


     0   1
   +---+---+
 0 | x | x |
   +---+---+
 1 | x | x |
   +---+---+

=> Felder sich bedrohender Damen:
[0][0]  und  [0][1]
[0][0]  und  [1][0]
[0][0]  und  [1][1]
[0][1]  und  [1][1]
[0][1]  und  [1][0]
[1][0]  und  [1][1]

#####################################################################*/
```

Zu 29.) Perfektes Mischen

```c
/*      perf_misch.c -- Matthias Jauernig, 11.12.03                          */
/*      Programm simuliert das perfekte Mischen eines Kartenstapels          */

#include <stdio.h>
#include <stdlib.h>

int main(void){
        // n,k,i sind die einzulesenden Werte
        // p,q,r,s sind Lauf- und Hilfsvariablen
        // anz gibt die minimale Durchlaufszahl bis zur Stapelwiederherstellung an
        int n, k, i, p, q, r, s, anz;
        // a ist das Array, was gemischt wird, b ist stets das Ausgangsarray
        // c und d sind die beiden Teilstapel, in die a immer zerlegt wird
        int *a, *b, *c, *d;

        printf( "--------------------------------\n"
                "| Perfektes Mischen von Karten |\n"
                "--------------------------------\n\n");
        //############################### Eingabe ####################################
        do{
                printf("Anzahl n der Karten (n gerade, n>0)?: ");
                scanf("%d", &n);
        }while((n<1 && printf("!! Anzahl muss größer 0 sein!\n\n"))
                || (n%2!=0 && printf("!! Anzahl muss gerade sein!\n\n")));
        do{
                printf("Wieviel mal soll perfekt gemischt werden (>0)?: ");
                scanf("%d", &k);
        }while(k<1 && printf("!! Anzahl muss größer 0 sein!\n\n"));
```

```
        do{
                printf("Wert i der i-ten Karte (0<=i<%d)?: ", n);
                scanf("%d", &i);
        }while((i<0 && printf("!! imuss größer gleich 0 sein!\n\n"))
                || (i>=n && printf("!! i muss kleiner als %d sein!\n\n", n)));

        //########################## Speicherreservierung ############################
        a=(int *)malloc(n*sizeof(int));
        b=(int *)malloc(n*sizeof(int));
        c=(int *)malloc(n/2*sizeof(int));
        d=(int *)malloc(n/2*sizeof(int));

        //########################### Arrays initialisieren ##########################
        for(p=0; p<n; p++)
                a[p]=b[p]=p;

        printf( "\n=> Nach 0 mal Mischen: Karte mit Wert %d"
                " ist an %d. Stelle im Stapel.\n", i, i);

        //########################## k-mal perfektes Mischen #########################
        // Schleife läuft solange, bis k-mal gemischt und der Ausgangsstapel wieder-
        // hergestellt wurde, also s==1 (s ist Hilfsvariable, die dies anzeigt)
        for(p=1, s=0; p<=k || s!=1; p++){
                // teile zunächst in die beiden Hilfsstapel
                for(q=0; q<n; q++)
                        if(q<n/2) c[q]=a[q];
                        else d[q-n/2]=a[q];
                // reißverschlussartig wieder zusammen setzen
                for(q=0, r=0; q<n; q+=2, r++){
                        a[q]=c[r];
                        a[q+1]=d[r];
                }
                // wenn k-mal gemischt, dann Position der Karte mit Wert i ausgeben
                if(p==k){
                        for(r=0, q=0; r<n && q!=1; r++)
                        if(a[r]==i){
                                printf( "=> Nach %d mal Mischen: Karte mit Wert %d "
                                        " ist an %d. Stelle im Stapel.\n\n", k, i, r);
                                q=1;
                        }
                }

                // wenn Ausgangsstapel noch nicht wieder hergestellt, dann schauen, ob
                // dies in diesem Durchgang der Fall ist, ansonsten s=0 belassen
                if(s!=1){
                        for(q=0, s=1; q<n && s==1; q++)
                                if(a[q]!=b[q]) s=0;
                        // Durchgangsanzahl in anz speichern
                        if(s==1) anz=p;
                }
        }
        printf( "==> Nach %d Durchläufen wurde der Ausgangsstapel"
                " wieder hergestellt!\n\n", anz);

        //############################## Aufräumen ###################################
        free(a); free(b); free(c); free(d);
        return 0;
}
```

```
/* 3 Testläufe
################################################################################
1.) wenn i=0, dann muss die Karte immer auf Platz 0 bleiben, egal wie groß n,k sind
################################################################################
-------------------------------
| Perfektes Mischen von Karten |
-------------------------------

Anzahl n der Karten (n gerade, n>0)?: 10000
Wieviel mal soll perfekt gemischt werden (>0)?: 10000
Wert i der i-ten Karte (0<=i<10000)?: 0

=> Nach 0 mal Mischen: Karte mit Wert 0 ist an 0. Stelle im Stapel.
=> Nach 10000 mal Mischen: Karte mit Wert 0  ist an 0. Stelle im Stapel.

==> Nach 300 Durchläufen wurde der Ausgangsstapel wieder hergestellt!

################################################################################
2.) wenn i=n-1, dann muss die Karte immer auf Platz n-1 sein, egal wie groß n,k sind
################################################################################
-------------------------------
| Perfektes Mischen von Karten |
-------------------------------

Anzahl n der Karten (n gerade, n>0)?: 1000
Wieviel mal soll perfekt gemischt werden (>0)?: 100000
Wert i der i-ten Karte (0<=i<1000)?: 999

=> Nach 0 mal Mischen: Karte mit Wert 999 ist an 999. Stelle im Stapel.
=> Nach 100000 mal Mischen: Karte mit Wert 999  ist an 999. Stelle im Stapel.

==> Nach 36 Durchläufen wurde der Ausgangsstapel wieder hergestellt!

################################################################################
3.) wenn i=4, n=10 und k=3, so muss i am Ende an 5. Stelle sein und nach 6 mal
Mischen der Ausgangsstapel wieder hergestellt werden (Rechnung per Hand)
################################################################################
-------------------------------
| Perfektes Mischen von Karten |
-------------------------------

Anzahl n der Karten (n gerade, n>0)?: 10
Wieviel mal soll perfekt gemischt werden (>0)?: 3
Wert i der i-ten Karte (0<=i<10)?: 4

=> Nach 0 mal Mischen: Karte mit Wert 4 ist an 4. Stelle im Stapel.
=> Nach 3 mal Mischen: Karte mit Wert 4  ist an 5. Stelle im Stapel.

==> Nach 6 Durchläufen wurde der Ausgangsstapel wieder hergestellt!
################################################################################*/
```

Zu 30.) GAUSS-Algorithmus ohne Pivotsuche

```c
/*      gaussalg.c -- Matthias Jauernig, 12.12.03                                */
/*      Programm berechnet die Lösungen eines LGS mit (n,n)-Koeffizientenmatrix   */

#include <stdio.h>
#include <stdlib.h>

/* -- printline() -- gibt eine Linie der Länge n mit dem Zeichen z aus ----------- */
void printline(int l, char z){
        int i;
        for(i=0; i<=l; i++)
                printf("%c",z);
        printf("\n");
}

/* -- printhead() -- gibt die Kopfzeile der Tabelle aus ------------------------- */
void printhead(int n){
        int i;
        char str[10];
        printf("\n");
        printline(10*(n+1), '=');
        for(i=0; i<n; i++){
                snprintf(str, 10, "x[%d]", i);
                printf("|%8s ", str);
        }
        printf("|      b    |\n");
        printline(10*(n+1), '=');
}

/* -- printgauss() -- gibt die aktuelle Koeffizientenmatrix a in Tabellenform aus - */
void printgauss(float **a, int n, char z){
        int i, j;
        for(i=0; i<n; i++){
                        for(j=0; j<=n; j++)
                                printf("| %7.2g ", a[i][j]);
                        printf("|\n");
                }
                printline(10*(n+1), z);
}

float absol(float z){
        if(z<0.0)       return -z;
        else            return z;
}

/* == main() ================================================================== */
int main(void){
        // **a: Koeffizientenmatrix; *x: Lösungsvektor
        // quot: Quotient a[i][k]/a[k][k]; sub: für Lösungsvektor zu berechnende Summe
        // i,j,k: Laufvariablen; n: Zahl für die (n,n)-Koeffizientenmatrix
        // flag: zeigt an, ob eine Zeile vor Ende des Algorithmus' komplett 0 ist
        float **a, *x, quot, sub;
        int i, j, k, n;
        char flag;

        //################################## Eingabe ###################################
        printf( "--------------------------------------\n"
                "| Gauss-Algorithmus ohne Pivot-Suche |\n"
                "--------------------------------------\n\n");
        do{
                printf("Wie groß ist n für die (n,n)-Koeffizientenmatrix?: ");
                scanf("%d", &n);
        }while(n<=0 && printf("!! n muss größer als 0 sein!\n\n"));

        // Speicher reservieren
```

```
a=(float **)malloc(n*sizeof(float *));
for(i=0; i<n; i++)
        a[i]=(float *)malloc((n+1)*sizeof(float));

printf("\nWerte der Koeffizientenmatrix eingeben:\n");
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                do{
                        printf("Wert von a[%d][%d] eingeben: ", i, j);
                        scanf("%f", &a[i][j]);
                }while( i==j && a[i][j]==0
                        && printf("!! a[i][i] darf nicht 0 sein!\n\n"));

printf("\nWerte des Vektors b eingeben:\n");
for(i=0; i<n; i++){
        printf("Wert von b[%d] eingeben: ", i);
        scanf("%f", &a[i][n]);
}

//######################### a auf Dreiecksform bringen ##########################
if(n<=6){
        printhead(n);
        printgauss(a, n, '~');
}

//1. Schleife: alle Zeilen von 0 bis n-1, höre auf, wenn eine Zeile komplett 0
//2. Schleife: subtrahiere von jeder Zeile unter k
//3. Schleife: subtrahiere von jeder Spalte
for(k=0, flag=0; k<n-1 && flag==0; k++){
        for(i=k+1; i<n; i++){
                quot=a[i][k]/a[k][k];
                for(j=k; j<=n; j++){
                        a[i][j]-=a[k][j]*quot;
                        //Fehler durch Rundung abfangen
                        if(absol(a[i][j])<1e-05) a[i][j]=0.0;
                }
        }
        if(n<=6) printgauss(a, n, (k==n-2) ? '=' : '-');

        //folgender Codeteil bewirkt, dass Ende, wenn eine Zeile 0 ist;
        //ist in dem Fall b[i]=0, so wird flag=1 gesetzt -> unendl. viele Lsg.
        //ist in dem Fall b[i]!=0, so wird flag=2 gesetzt -> keine Lösungen
        for(i=k+1; i<n && flag!=2; i++){
                for(j=k, flag=1; j<n && flag==1; j++)
                        if(a[i][j]!=0) flag=0;
                if(flag==1)
                        if(a[i][n]!=0)
                                flag=2;
        }
        if(flag!=0){
                if(flag==1)
                        printf( "=> %d. Zeile 0=0: also unendlich "
                                "viele Lösungen!\n", i);
                else
                        printf( "=> %d. Zeile 0!=%g: "
                                "also keine Lösung!\n", i, a[i-1][n]);
        }

}


//#################### durch Rücksubstitution x[i] berechnen ###################
if(flag==0){
        x=(float *)malloc(n*sizeof(float));
        printf("\nLösungen nach dem Gaussschen Algorithmus:\n");
        for(i=n-1; i>=0; i--){
                sub=0;
```

```
                        //Summe berechnen, die abgezogen wird
                        for(j=i+1; j<n; j++)
                                sub+=a[i][j]*x[j];
                        x[i]=(a[i][n]-sub)/a[i][i];
                        printf("=> x[%d] = %g\n", i, x[i]);
                }
                free(x);
        }
        printf("\n\n");

        //################################## Aufräumen ###################################
        for(i=0; i<n; i++)
                free(a[i]);
        free(a);
        return 0;
}

/* Beispielabläufe
################################################################################################
--------------------------------------
| Gauss-Algorithmus ohne Pivot-Suche |
--------------------------------------

Wie groß ist n für die (n,n)-Koeffizientenmatrix?: 3

Werte der Koeffizientenmatrix eingeben:
Wert von a[0][0] eingeben: 1
Wert von a[0][1] eingeben: 1
Wert von a[0][2] eingeben: 1
Wert von a[1][0] eingeben: 2
Wert von a[1][1] eingeben: 2
Wert von a[1][2] eingeben: 2
Wert von a[2][0] eingeben: 1
Wert von a[2][1] eingeben: 1
Wert von a[2][2] eingeben: 1

Werte des Vektors b eingeben:
Wert von b[0] eingeben: 1
Wert von b[1] eingeben: 1
Wert von b[2] eingeben: 1

=========================================
|    x[0] |    x[1] |    x[2] |     b   |
=========================================
|       1 |       1 |       1 |       1 |
|       2 |       2 |       2 |       1 |
|       1 |       1 |       1 |       1 |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
|       1 |       1 |       1 |       1 |
|       0 |       0 |       0 |      -1 |
|       0 |       0 |       0 |       0 |
-----------------------------------------
=> 2. Zeile 0!=-1: also keine Lösung!

################################################################################################
--------------------------------------
| Gauss-Algorithmus ohne Pivot-Suche |
--------------------------------------

Wie groß ist n für die (n,n)-Koeffizientenmatrix?: 3

Werte der Koeffizientenmatrix eingeben:
Wert von a[0][0] eingeben: 1
Wert von a[0][1] eingeben: 1
Wert von a[0][2] eingeben: 1
Wert von a[1][0] eingeben: 1
Wert von a[1][1] eingeben: 1
```

```
Wert von a[1][2] eingeben: 1
Wert von a[2][0] eingeben: 1
Wert von a[2][1] eingeben: 1
Wert von a[2][2] eingeben: 1

Werte des Vektors b eingeben:
Wert von b[0] eingeben: 1
Wert von b[1] eingeben: 1
Wert von b[2] eingeben: 1

=========================================
|    x[0] |    x[1] |    x[2] |     b   |
=========================================
|       1 |       1 |       1 |       1 |
|       1 |       1 |       1 |       1 |
|       1 |       1 |       1 |       1 |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
|       1 |       1 |       1 |       1 |
|       0 |       0 |       0 |       0 |
|       0 |       0 |       0 |       0 |
-----------------------------------------
=> 3. Zeile 0=0: also unendlich viele Lösungen!

################################################################################################
-------------------------------------
| Gauss-Algorithmus ohne Pivot-Suche |
-------------------------------------

Wie groß ist n für die (n,n)-Koeffizientenmatrix?: 2

Werte der Koeffizientenmatrix eingeben:
Wert von a[0][0] eingeben: 5
Wert von a[0][1] eingeben: 3
Wert von a[1][0] eingeben: 4
Wert von a[1][1] eingeben: 6

Werte des Vektors b eingeben:
Wert von b[0] eingeben: 2
Wert von b[1] eingeben: 6

================================
|    x[0] |    x[1] |     b   |
================================
|       5 |       3 |       2 |
|       4 |       6 |       6 |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
|       5 |       3 |       2 |
|       0 |     3.6 |     4.4 |
================================

Lösungen nach dem Gaussschen Algorithmus:
=> x[1] = 1.22222
=> x[0] = -0.333333                                                                    */
```