

```
/******1. Beispiel: Zahl-->Zahl *****
```

Die Funktion "funktion\_zahl()" gibt das Quadrat einer Zahl aus.

Aufruf des Programms: ./zahlrein\_zahlraus x

"x" ist eine beliebige Zahl, die durch ein Leerzeichen getrennt hinter dem Programmnamen eingegeben wird.

```
*****/
```

```
#include<stdio.h>
```

```
double funktion_zahl(double zahl_2)          /* 1) */
{
    double zahl_1;
    zahl_1 = zahl_2*zahl_2;
    return zahl_1;
}
```

```
int main(int argc, char*argv[])             /* 2) */
{
    if (argc!=2)                             /* 3) */
    {
        printf("Programmaufruf ist falsch.\n");
        printf("Eingabe: %s <x>\n", argv[0]);
        exit(1);
    }
```

```
    double x = atof(argv[1]);                /* 4) */

    printf("Ergebnis: %f\n", funktion_zahl(x));
return 0;
}
```

```
***** Erläuterungen*****
```

1) Unsere Funktion "funktion\_zahl()" bekommt einen double-Wert als Argumente zugeteilt.

Der Rückgabewert ist wieder ein double-Wert.

2) "argc" und "argv" erlauben es beim Aufruf des Programms Zeichenfolgen als Parameter zu übergeben.

argc (argument count) ist ein int-Parameter, der die Anzahl der beim Programmaufruf angegebenen Zeichenfolgen enthält (inclusive des Programmnamens selbst)

argv (argument vector) ist ein Vektor von Zeigern auf die im Programmaufruf angegebenen Zeichenfolgen.

argv[0] zeigt auf den Programmnamen selbst, argv[1] auf den ersten Parameter usw.

3) Überprüfung der Anzahl der Programmaufrufparameter. Ist diese falsch, dann entsteht eine Speicherschutzverletzung von Typ double.

4) "atof" wandelt eine Zeichenfolge vom Typ char in eine double-Zahl um.

```
*****/
```

/\*\*\*\*\*\*2. Beispiel: Matrix-->Zahl \*\*\*\*\*

Die Funktion "funktion\_zahl()" summiert die Vektorkoeffizienten auf. Das Ergebnis wird durch printf in der Konsole ausgegeben.

Aufruf des Programms: ./mateixrein\_zahlraus N datei.txt

"datei.txt" ist die Textdatei, welche die durch Leerzeichen getrennte Vektorkoeffizienten beinhaltet.

"N" wird als die Dimension des Vektors interpretiert.

\*\*\*\*\*/

```
#include<stdio.h>
#include<stdlib.h>                                /* 1) */

double funktion_zahl(double *vektor, int N)      /* 2) */
{
    double zahl=0.;

    for (int i=0; i<N; i++)
    {
        zahl += vektor[i];
    }
    return zahl;
}

int main(int argc, char *argv[])
{
    if (argc!=3)
    {
        printf("Programmaufruf ist falsch.\n");
        printf("Eingabe: %s <N> <dateiname>\n", argv[0]);
        exit(1);
    }

    int dim = atoi(argv[1]);                       /* 3) */

    FILE*f;                                       /* 4) */
    char *filename = argv[2];
    f = fopen(filename, "r");

    if(!fopen(filename, "r"))                     /* 5) */
    {
        printf("Datei existiert nicht!!\n");
        exit (1);
    }

    double *v = (double*) malloc(sizeof(double)*dim); /* 6) */

    for(int i=0; i<dim; i++)
    {
        fscanf(f, "%lf", &v[i]);                 /* 7) */
    }
}
```

```

    printf("Summe der Vektorkoeffizienten: %f\n", funktion_zahl(v,dim)); /*
8) */

free (v); /* 9) */

return 0;
}

/***** Erläuterungen *****/

1) Die Bibliothek "stdlib" wird eingebunden, da sie die Funktionen "atoi" und
"malloc" enthält.

2) Unsere Funktion "funktion_zahl(...)" bekommt einen Vektor, und dessen
Dimension als Argumente zugeteilt.
Der Rückgabewert ist eine double-Zahl.

3) "atoi" wandelt eine Zeichenfolge vom Typ char in eine int-Zahl um.

4) "FILE *f ... f=fopen(filename, "r");" - eine Datei mit dem Namen
"filename" wird zum lesen "r" geöffnet.

5) Hier wird überprüft, ob die einzulesende Datei im Verzeichnis existiert.

6) "malloc(sizeof...)" - diese Funktion reserviert einen Speicherbereich der
Größe sizeof... .

7) "fscanf(..., "%lf", &...)" liest Variablen aus einer Datei. %lf für
&"double-Zahl" und %d für &"int-Zahl"

8) Unsere Funktion "funktion_zahl(...)" wird in der Mainfunktion aufgerufen.

9) "free(v)" gibt den Speicherbereich, auf den v Zeigt und der durch die
Funktion malloc reserviert wurde,
wieder frei.

*****/

```

*/\*\*\*\*\*\*3. Beispiel: Matrix-->Zahl \*\*\*\*\*\*/*

Die Funktion "funktion\_zahl()" summiert die Koeffizienten einer gegebenen Matrix auf.

Aufruf des Programms: `./mateixrein_zahlraus N M datei.txt`

"datei.txt" ist die Textdatei, welche die durch Leerzeichen getrennte Matrixkoeffizienten beinhaltet.

"N" wird als die Anzahl der Zeilen und "M" als die Anzahl der Spalten dieser Matrix interpretiert.

*\*\*\*\*\*/*

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

double funktion_zahl(double **matrix, int n, int m)    /* 1) */
{
    double zahl=0.;

    for (int i=0; i<n; i++)
    {
        for(int j=0; j<m; j++)
        {
            zahl = zahl + matrix[i][j];
        }
    }
    return zahl;
}

int main(int argc, char*argv[])
{
    if (argc!=4)
    {
        printf("Programmaufruf ist falsch.\n");
        printf("Eingabe: %s <N> <M> <dateiname>\n", argv[0]);
        exit(1);
    }

    int i,j;
    int N = atoi(argv[1]);
    int M = atoi(argv[2]);
    printf("N = %i\n", N);
    printf("M = %i\n", M);

    FILE *f;
    char *filename = argv[3];
    f = fopen(filename, "r");

    if(!fopen(filename, "r"))
    {
        printf("Datei existiert nicht!!\n");
        exit (1);
    }
}
```

```

double **A = (double**) malloc (sizeof(double)*N);      /* 2) */
for (i=0; i<N; i++)
{
    A[i] = (double*) malloc (sizeof(double)*M);
}

for (i=0;i<N;i++)
{
    for (j=0; j<M; j++)
    {
        fscanf(f, "%lf", &A[i][j]);
    }
}

printf("Summe der Matrixkoeffizienten: %f\n", funktion_zahl(A,N,M));

free (A);

return 0;

}

/*****Erläuterungen*****/

1)      Unsere Funktion "funktion_zahl(...)" bekommt eine Matrix, und zwei
integer-Werte als Argumente zugeteilt.
        Der Rückgabewert ist eine double-Zahl.

2)      Hier wird Speicher für einen doppelten array freigegeben. Vom Prinzip her
wie im 2. Beispiel.

*****/

```

```
/******4. Beispiel: Zahl-->Vektor *****/
```

Die Funktion "funktion\_vektor()" gibt einen Vektor einer beliebigen Länge aus, dessen Koeffizienten die Erhöhung um Eins einer gegebenen Zahl ist.

Aufruf des Programms: ./zahlrein\_vektorraus N x

"N" ist die gewünschte Dimension des Vektors und "x" ist eine beliebige einzugebene Zahl.

```
*****/
```

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

```
double *funktion_vektor(double zahl, int N )          /* 1) */
{
    double *v = (double*) malloc(sizeof(double)*N);
    for(int i=0; i<N; i++)
    {
        v[i] = zahl;
        zahl++;
    }
    return v;
}
```

```
int main(int argc, char*argv[])
{
    if (argc!=3)
    {
        printf("Programmaufruf ist falsch.\n");
        printf("Eingabe: %s <N> <x>\n", argv[0]);
        exit(1);
    }

    double x =atof(argv[1]);          /* 2) */
    int dim = atoi(argv[2]);

    for(int i=0; i<dim; i++)
    {
        printf( "%f\n", funktion_vektor(x,dim) [i] );
    }

    return 0;
}
```

```
***** Erläuterungen *****/
```

1) Unsere Funktion "funktion\_vektor(...)" bekommt eine double-Zahl, und eine integer-Zahl als Argumente zugeteilt. Der Rückgabewert ist ein array der Länge "dim".

2) "atof" wandelt eine Zeichenfolge vom Typ char in eine double-Zahl um.

```
*****/
```

/\*\*\*\*\*\*5. Beispiel: Vektor --> Vektor \*\*\*\*\*/

Die Funktion "funktion\_vektor()" addiert zwei in einer Textdatei (tabelle.txt) enthaltene Vektoren der gleichen Länge N.

Diesmal gibt das Programm das Ergebnis in der Textdatei beispiel\_5.txt aus.

Aufruf des Programms: ./vektorrein\_vektorraus N tabelle.txt

Die Datei tabelle.txt sollte so aussehen:

```
x_1    y_1
x_2    y_2
...    ...
...    ...
x_N    x_N
```

\*\*\*\*\*/

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

```
double *funktion_vektor(double *vec_1, double *vec_2, int N )      /* 1) */
{
```

```
    double *v = (double*) malloc(sizeof(double)*N);
```

```
    for(int i=0; i<N; i++)
```

```
    {
        v[i] = vec_1[i] + vec_2[i];
    }
```

```
    return v;
```

```
}
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    if (argc!=3)
```

```
    {
```

```
        printf("Programmaufruf ist falsch.\n");
```

```
        printf("Eingabe: %s <N> <dateiname>\n", argv[0]);
```

```
        exit(1);
```

```
    }
```

```
    int i;
```

```
    int dim = atoi(argv[1]);
```

```
    FILE *fr;
```

```
        /* 2) */
```

```
    FILE *fp;
```

```
    char * filename = argv[2];
```

```
    fr = fopen(filename, "r");
```

```
    fp = fopen("beispiel_5.txt", "w");
```

```
    if(!fopen(filename, "r"))
```

```
    {
```

```
        printf("Datei existiert nicht!!\n");
```

```
        exit (1);
```

```
    }
```

```
    double *a = (double*) malloc(sizeof(double)*dim);
```

```

double *b = (double*) malloc(sizeof(double)*dim);

for (i=0; i<dim; i++)
{
    fscanf(fr,"%lf          %lf\n", &a[i], &b[i]);
}

for (i=0; i<dim; i++)
{
    fprintf(fp,"%f \n", funktion_vektor(a,b,dim)[i]);
}
fclose(fp);

printf("\n Schau mal in der Datei beispiel_5.txt nach.\n");

free (a);
free (b);
return 0;
}

```

*/\*\*\*\*\* Erläuterungen \*\*\*\*\*/*

1) Unsere Funktion "funktion\_vektor(...)" bekommt zwei arrays und eine integer-Zahl als Argumente zugeteilt.

Der Rückgabewert ist ein wieder ein array der Länge "dim".

2) Das Pondon zum Einlesen "fr=fopen(filename,"r"); mit "fscanf(fr,"%lf\n",&..);" einer Textdatei ist die Ausgabe in eine Textdatei.

"fp = fopen("beispiel\_5.txt", "w");" mit "fprintf(fp,...)". Natürlich braucht die Datei einen Namen, z.B. "beispiel\_5.txt". Außerdem muß die Funktion fopen auch wissen, was sie tun soll, nämlich Schreiben: "w"

Falls schon eine Datei mit demselben Namen (auch "beispiel\_5.txt") existiert, so wird ihr Inhalt gelöscht und neu überschrieben.

"fclose(fp)" schließt die Datei wieder.

TIPP:

Es gibt 2 Editoren, die für die schnelle Einsicht und das schnelle Ändern einer Textdatei sehr praktisch sind:

vim und emacs

Konsole: vim 'dateiname'

i -> INSERT-MODUS

entf -> Zeichen entfernen

esc -> COMMANDO-MODUS

Wenn man wieder raus will: (COMMANDO-MODUS):

strg + q, dann "g!" eingeben -> beenden ohne zu sichern

strg + q, dann "g:" eingeben -> beenden mit sichern

Konsole: emacs 'dateiname'

ändere deine Daten, wie Du magst

strg + s -> speichern

strg + x, dann strg + c, dann y oder n -> beenden

*\*\*\*\*\*/*

/\*\*\*\*\*\*6. Beispiel: Matrix --> Vektor \*\*\*\*\*/

Die Funktion "funktion\_vektor()" kopiert die Diagonalelemente einer Matrix A in einen array.

Die Matrix befindet sich in einer Textdatei , z.B."matrix.txt" und ist diesmal quadratisch mit Dimension N.

Das Programm gibt das Ergebnis wieder in einer Textdatei mit einem beliebig einzugebenen Namen aus.

(Am besten eine geläufige Endung wie \*.txt oder \*.dat dafür verwenden, sonst kriegen die Editoren Probleme.)

Aufruf des Programms: ./matrixrein\_vektorraus N matrix.txt neuerdateiname.txt

\*\*\*\*\*/

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

double *funktion_vektor(double **A, int N )
{
    double *v = (double*) malloc(sizeof(double)*N);

    for (int i=0; i<N; i++)
    {
        v[i] = A[i][i];
    }
    return v;
}

int main(int argc, char*argv[])
{
    if (argc!=4)
    {
        printf("Programmaufruf ist falsch.\n");
        printf("Eingabe: %s <N> <matrixdatei> <neuer dateiname>\n",
argv[0]);
        exit(1);
    }

    int i,j;
    int N = atoi(argv[1]);

    FILE *fr,*fp;
    char *filename = argv[2];
    char *newfilename = argv[3];
    fr = fopen(filename, "r");
    fp = fopen(newfilename, "w");

    if(!fopen(filename, "r"))
    {
        printf("Datei existiert nicht!!\n");
        exit (1);
    }

    double **A = (double**) malloc (sizeof(double)*N);
    for (i=0; i<N; i++)
    {
        A[i] = (double*) malloc (sizeof(double)*N);
```

```
}  
  
for (i=0;i<N;i++)  
{  
    for (j=0; j<N; j++)  
    {  
        fscanf(fr, "%lf", &A[i][j]);  
    }  
}  
  
for (i=0; i<N; i++)  
{  
    fprintf(fp, "%f\n", funktion_vektor(A,N)[i]);  
}  
fclose(fp);  
  
free (A);  
  
return 0;  
}
```

/\*\*\*\*\*\*7. Beispiel: Zahl --> Matrix \*\*\*\*\*/

Die Funktion "funktion\_matrix()" hat zwei int-Werte als Argumente und gibt eine Matrix zurück.

Eine Matrix mit Einträgen nach dem Telefontasten-Prinzip wird ab einem gewünschten int-Wert bis zur gewünschten Dimension erstellt und auf der Konsole ausgegeben.

Aufruf des Programms: ./zahlrein\_matrixraus "startwert" "dim"

\*\*\*\*\*/

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

int **funktion_matrix(int zahl, int dim)
{
    int i, j;
    int **A = (int**) malloc(sizeof(int*)*dim);          /* 1) */
    for(i=0; i<dim; i++)
        A[i] = (int*) malloc(sizeof(int)*dim);

    for(i=0; i<dim; i++)
    {
        for(j=0; j<dim; j++)
        {
            A[i][j] = zahl++;
        }
    }
    return A;
}

int main (int argc, char *argv[])
{
    if (argc!=3)
    {
        printf("Programmaufruf ist falsch.\n");
        printf("Eingabe: %s <startwert> <dim>\n", argv[0]);
        exit(1);
    }

    int i, j;
    int x = atoi(argv[1]);
    int N = atoi(argv[2]);

    for(i=0; i<N; i++)
    {
        for(j=0; j<N; j++)
        {
            printf(" %3d ", funktion_matrix(x,N)[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

*/\*\*\*\*\*\* Erläuterungen\*\*\*\*\**

1) `sizeof(int)>>sizeof(int*)`

*Die letzten beiden Fälle `vektorrein_matrixraus` und `matrixrein_matrixraus` werden nicht mehr weiter aufgeführt.*

*\*\*\*\*\*/*

*\*\*\*\*\*8. Beispiel: Auswahl einer Funktion von Mehreren\*\*\*\*\**

*Gegeben sind vier einfache Funktionen, wovon das Hauptprogramm aber nur eine aufrufen soll.*

*Die ausgewählte Funktion wird in einem beliebigen Intervall [a,b] mit Schrittweite 0.1 ausgewertet.*

*Was man aber von diesem Beispiel lernen soll, ist wie man mit einer Zeichenkette eine den vier Funktionen aufruft.*

*Die erste funktion trägt z.B. den Namen "claudia", die zweite "dieter", usw..*

*Zuletzt wird noch die Anzahl der Funktionsauswertungen bestimmt, wobei der Unterschied einer lokalen und einer globalen Variablen deutlich wird.*

*Aufruf des Programms: ./choose a b "name\_der\_Funktion"*

*\*\*\*\*\*/*

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
```

```
int k=0;          /* 1) */
```

```
double claudia(double x)
{
    k++;
    return x;
}
```

```
double dieter(double x)
{
    k++;
    return x*x;
}
```

```
double manfred(double x)
{
    k++;
    return pow(x,3);
}
```

```
double sabine(double x)
{
    k++;
    return pow(x,4);
}
```

```
int main(int argc, char*argv[])
{
    if (argc!=4)
    {
        printf("Programmaufruf ist falsch.\n");
    }
}
```

```

        printf("Eingabe: %s <untere Grenze> <obere Grenze> <Name der
Funktion>\n", argv[0]);
        exit(1);
    }

    double a = atof(argv[1]);
    double b = atof(argv[2]);
    double x, f;
    int auswahl;
    /* 2) */

    if(0==strcmp(argv[3], "claudia"))
        auswahl = 1;
    /* 3) */

    else if (0==strcmp(argv[3], "dieter"))
        auswahl = 2;

    else if (0==strcmp(argv[3], "manfred"))
        auswahl = 3;

    else if (0==strcmp(argv[3], "sabine"))
        auswahl = 4;

    else
    {
        printf("falsche Eingabe\n");
        exit(1);
    }

    for (x=a; x<=(b+0.1); x=x+0.1)
    {
        switch (auswahl)
        {
            /* 4) */
            case 1:
                f = claudia(x);
                break;

            case 2:
                f = dieter(x);
                break;

            case 3:
                f = manfred(x);
                break;

            case 4:
                f = sabine(x);
                break;

            /*default: printf("Die genannte Funktion ist nicht vorhanden.\n");*/
        }

        printf("x = %5.5f\tf(x) = %5.5f\n", x, f);
    }
    printf("# der Fktauswertungen: %i\n", k);

    k=0;

    return 0;
}

```

*/\*\*\*\*\*\* Erläuterungen\*\*\*\*\**

1) *"int k" ist in diesem Beispiel eine globale Variable, sie wird außerhalb des Hauptprogramms deklariert.*

2) *"double f,x" sind z.B. lokale Variablen, sie können nur im Hauptprogramm benutzt werden.*

3) *"strcmp(..., ...)" Vergleicht zwei Zeichenketten miteinander und gibt folgende Werte zurück:*

*<0 <=> "Zeichenkette1" < "Zeichenkette2"*

*=0 <=> "Zeichenkette1" = "Zeichenkette2"*

*>0 <=> "Zeichenkette1" > "Zeichenkette2"*

*Man muß string.h einbinden!!*

4) *Wie die switch-Funktion genau funktioniert, kann man dem Beispiel hoffentlich gut entnehmen. Falls keine*

*case-Auswahl zutrifft, kann man eine Alternative einführen: "default: Anweisung;". Da wir in unserem Beispiel*

*aber schon eine Abbruchsbedingung implementiert haben macht "default:" hier keinen Sinn.*

*Ich habe es deshalb auskommentiert.*

*\*\*\*\*\*/*