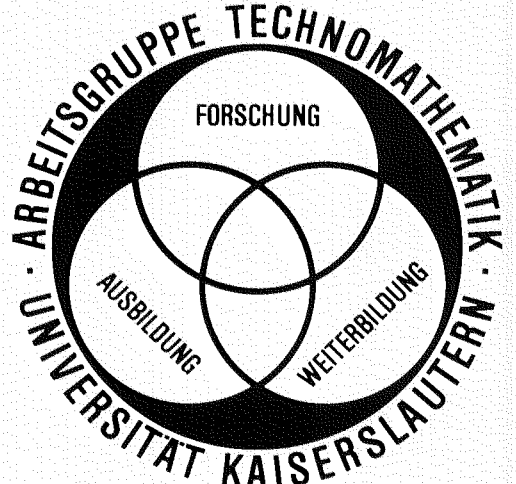
BERICHTE
DER
ARBEITSGRUPPE TECHNOMATHEMATIK

FORSCHUNG - AUSBILDUNG - WEITERBILDUNG

THE C PROGRAMMES FOR "NUMERICAL METHODS (PROGRAMMES AND IMPLEMENTATION)"

Michael Schreiner

Bericht 96 - 160 März 1996



FACHBEREICH MATHEMATIK

The C Programmes for "Numerical Methods (Programmes and Implementation)"

Michael Schreiner
University of Kaiserslautern
Laboratory of Technomathematics
Geomathematics Group
P.O. Box 30 49
67653 Kaiserslautern
Germany

e–mail: schreiner@mathematik.uni–kl.de

1 Preface

These lecture notes result from the lecture "Numerical Methods (Programmes and Implementation)" held by the author in the winter term 1995/1996 in the postgraduate programme "Mathematics for Industry". One of the main objectives of this lecture was to present in a beginner's course an introduction to the C programming language and to enable the students to apply C in relevant problems arising in industrial mathematics.

The motivation for writing these notes is the fact, that C — as well as any other programming language — can only be learned by studying many example programmes and by doing much programming work. In these notes, more than 40 programmes showing different levels of difficulty are collected. They are more or less usefull and demonstrate the most elements of C. Particular emphasis is drawn to avoid the presentation of programme fragments. Each programme is complete and should work correctly. Only those elements of C are used within the examples, that were already known at the actual stage of the course. Since this report is designed as an accompanying tool for the students following the lecture, comments are used only rarely.

One of the most important characteristics of the C programming language is the intensive use of pointers. They allow the formulation of very short and clear programmes (at least if one has some practice in C). As an example consider the problem of copying one string (vector of characters) to another. This could be implemented as

```
void strcpy(char s[], char t[])
{
   int i;

   i = 0;
   while (t[i] != '\0') {
      s[i] = t[i];
      i = i + 1;
      }
   s[i] = '\0';
}
```

An experienced C–programmer would probably prefer an implementation like

```
void strcpy(char s[], char t[])
{
   while (*s++ = *t++)
      ;
}
```

Unfortunately, the use of pointers usually causes many difficulties for the beginner in C. Therefore, a large part of these notes is dedicated to clarify the concept of pointers.

The outline of these notes follow the structure of the lecture:

Many books on C are available. We mention here only the standard book on C:

Brian W. Kernighan, Dennis M. Ritchie: The C Programming Language, Second Edition, ANSI C, Prentice–Hall, Englewood Cliffs, New Jersey, 1983

A german translation is available:

Brian W. Kernighan, Dennis M. Ritchie: Programmieren in C, 2. Ausgabe, ANSI C, Hanser, München, 1990

2 Introduction to C and Simple Examples

```
/* Program 2_1 */
/* This is a comment */

#include <stdio.h>

void main()
{
   printf("Hello, world\n");
}
```

```
/* Program 2_2 */

#include <stdio.h>

void main()
{
   printf("Hello");
   printf(", world");
   printf("\n");
}
```

```
/* Program 2_3 */

#include <stdio.h>

void main()
{

/* declaration of variables */

   int fahr, celsius;
   int lower, upper, step;

/* initialisation */

   lower = 0;
   upper = 300;
   step = 20;
```

```
   fahr = lower;

   while (fahr <= upper)
   {
      celsius = 5 * (fahr - 32) /9;
      printf("%6d %6d\n", fahr, celsius);
      fahr = fahr + step;
   }
}
```

```
/* Program 2_4 */

#include <stdio.h>

void main()
{

/* declaration of variables */

   int fahr, celsius;
   int lower, upper, step;

/* initialisation */

   lower = 0;
   upper = 300;
   step = 20;

   fahr = lower;

   while (fahr <= upper)
   {
      celsius = 5 * (fahr - 32) /9;
      printf("%6d %6d\n", fahr, celsius);
      fahr = fahr + step;
   }
}
```

```
/* Program 2_5 */

#include <stdio.h>

void main()
{

/* declaration of variables */

   float fahr, celsius;
   int lower, upper, step;

/* initialisation */

   lower = 0;
   upper = 300;
   step = 20;

   fahr = lower; /* be careful! fahr is  float, lower is integer */

   while (fahr <= upper)
   {
      celsius = (5.0/9.0) * (fahr - 32.0);
      printf("%3.0f %6.1f\n", fahr, celsius);
      fahr = fahr + step;
   }
}
```

```
/* Program 2_6 */

#include <stdio.h>


void main()
{


   int  fahr;

   for (fahr=0; fahr <= 300; fahr = fahr + 20)
         printf("%3d %6.1f\n", fahr, (5.0/9.0) * (fahr - 32));
}
```

```c
/* Program 2_7 */

#include <stdio.h>

/* definition of constants */

#define LOWER 0
#define UPPER 300
#define STEP 20


void main()
{


   int  fahr;

   for (fahr=LOWER; fahr <= UPPER; fahr = fahr + STEP)
         printf("%3d %6.1f\n", fahr, (5.0/9.0) * (fahr - 32));
}
```

```c
/* Program 2_8 */

#include <stdio.h>
#include <math.h>

#define PI 3.1415927


void main()
{

   float v[2], v_rot[2];
   float phi;
   float A[2][2];


   scanf("%f", &phi); /* input of angle phi */
   scanf("%f %f", &(v[0]), &(v[1]));  /* input of vector v */

   phi = PI * phi/180.0;

   A[0][0] = A[1][1] = cos(phi);
   A[0][1] = sin(phi);
   A[1][0] = -A[0][1];
```

```
    v_rot[0] = A[0][0]*v[0] + A[0][1]*v[1];
    v_rot[1] = A[1][0]*v[0] + A[1][1]*v[1];

    printf("%6.2f  %6.2f\n",v_rot[0], v_rot[1]);
}
```

```
/* Program 2_9 */

#include <stdio.h>


/* declaration of function power */

int power(int base, int n);

/* main */

void main()
{
    int i;

    for (i=0; i < 10; i++)
        printf("%5d %7d %7d\n", i, power(2,i), power(3,i));
}

/* definition of function power */

int power(int base, int n)
{
    int i,p;

    p = 1;
    for (i=1; i <= n; i++)
        p = p * base;

    return p;
}
```

```
/* Program 2_10 */

#include <stdio.h>


/* declaration of function power */

int power(int base, int n);

/* main */

void main()
{
   int i;

   for (i=0; i < 10; i++)
      printf("%5d %7d %7d\n", i, power(2,i), power(3,i));
}

/* definition of function power */

int power(int base, int n)
{
   int p;

   for (p=1; n > 0; n--)
      p = p * base;

   return p;
}
```

```
/* Program 2_11 */

#include <stdio.h>


/* declaration of functions */

int length_of_string(char buf[]);


/* main */

void main()
{
   char string1[80], string2[80];
   int i,l;

   scanf("%s",string1);

   l = length_of_string(string1);


   for (i=0; i < l; i++)
      string2[i] = string1[l-i-1];

   string2[l] = '\0';

   printf("%s %s\n", string1, string2);

}

/* definition of function length_of_string */

int length_of_string(char buf[])
{
   int l;

   l=0;
   while (buf[l] != '\0')
      l++;

   return l;
}
```

3 Data Types, Operators

```
/* Program 3_1 */


#include <stdio.h>

void squeeze(char s[], char c);


void main()
{

   char c, s[80];


   scanf("%s\n",s);
   scanf("%c", &c);

   squeeze(s,c);

   printf("%s\n",s);
}


/* Delete all characters c in s */

void squeeze(char s[], char c)
{
   int i,j;

   for (i=j=0; s[i] != '\0'; i++)
      if (s[i] != c)
         s[j++] = s[i];
   s[j] = '\0';
}
```

```
/* Program 3_2 */

#include <stdio.h>

void cat(char s[], char t[]);


void main()
{

   char s[80], t[80];


   scanf("%s\n",s);
   scanf("%s",t);

   cat(s,t);

   printf("%s\n",s);
}


/* Concatenate t on s (assuming s is large enough) */

void cat(char s[], char t[])
{
   int i,j;

   i=j=0;

   while (s[i] != '\0')     /* Find the end of s */
      i++;

   while ((s[i++] = t[j++]) != '\0')     /* Copy t */
      ;
}
```

---

4 Control Structures

```
/* Program 4_1 */

/* bisection */



#include <stdio.h>
#include <math.h>


#define abs(x) ((x) > 0 ? (x) : -(x))

#define EPS 1e-7
#define TRUE_SOLUTION 0.7390851332151

double f(double x);

int bisection(double a, double b, double *result);



void main()
{

   double left, right, solution;

   printf("Left: ");
   scanf("%lf", &left);
   printf("Right: ");
   scanf("%lf", &right);

   if (bisection(left, right, &solution))
      printf("Solution: %20.12lf\n", solution);
   else
      printf("Bisection failed!\n");
}


double f(double x)
   {
   return cos(x)-x;
   }
```

```
int bisection(double left, double right, double *result)
{
   double mid;
   double left_value, right_value, mid_value;

/* Check if f(a)*f(b) < 0 */

   left_value = f(left);
   right_value = f(right);

   if (left_value*right_value > 0.)
      return 0;

   printf("Iteration x_value                f(x)_value              error\n");
   while(abs(left_value-right_value) > EPS)
      {
      mid = (left+right)/2;
      mid_value = f(mid);

      printf("          %20.14lf %20.14lf %20.14lf\n",
               mid, mid_value, mid - TRUE_SOLUTION);


      if (left_value*mid_value <0.)
         {
         right = mid;
         right_value = mid_value;
         }
      else
         {
         left = mid;
         left_value = mid_value;
         }
      }

   *result = (left + right)/2.;

   return 1;
}
```

```
/* Program 4_2 */

/* Count digits, white spaces, and others */


#include <stdio.h>

void main()
{

   int c, i, nwhite, nother, ndigit[10];


/* initialisation */

   nwhite = nother = 0;
   for (i=0; i<10; i++)
      ndigit[i] = 0;

   while ((c= getchar()) != EOF)
      {
      printf("%c",c);
      switch (c)
         {
         case '0': case '1': case '2': case '3':
         case '4': case '5': case '6': case '7':
         case '8': case '9':
            ndigit[c-'0']++;
            break;
         case ' ':
         case '\n':
         case '\t':
            nwhite++;
            break;
         default:
            nother++;
            break;
         }
      }

   printf("Digits:\n");
   for (i=0; i<10; i++)
      printf ("%2d:%4d\n", i, ndigit[i]);
   printf("white: %4d\n", nwhite);
   printf("other: %4d\n", nother);
}
```

```
/* Program 4_3 */

/* Newton */

#include <stdio.h>
#include <math.h>

#define abs(x) ((x) > 0 ? (x) : -(x))

#define EPS 1e-7
#define NEARLY_ZERO 1e-12
#define MAX_ITERATIONS 100
#define TRUE_SOLUTION 0.7390851332151

double f(double x);
double f1(double x);
int Newton(double x, double *result, int *error_type);

void main()
{

   double x, solution;
   int error_type;

   printf("Initial value: ");
   scanf("%lf", &x);

   if (Newton(x, &solution, &error_type))
      printf("Solution: %20.12lf\n", solution);
   else
      switch (error_type) {
         case 1:
            printf("Error! Gradient was zero!\n");
            break;
         case 2:
            printf("Error! Method did not converge!\n");
            break;
         }

}


double f(double x)
   {
   return cos(x)-x;
   }

double f1(double x)
   {
   return -sin(x) -1.;
   }
```

```
int Newton(double x, double *result, int *error_type)
{

   double current_value, gradient;
   double old_x;
   int    count = 1;

   printf("Iteration x_value                f(x)_value              error\n");


   current_value = f(x);

   do
      {

      printf("%10d %20.14lf %20.14lf %20.14lf\n",
               count, x, current_value, x - TRUE_SOLUTION);


      if (abs(gradient= f1(x)) < NEARLY_ZERO)
         {
         *error_type = 1;
         return 0;
         }
      else
         {
         old_x = x;
         x = x - current_value/gradient;
         current_value = f(x);
         }
      }
   while (abs(old_x-x) > EPS && count++ < MAX_ITERATIONS);

   if (abs(old_x-x) <= EPS) {
      *result = x;
      return 1;
      }
   else
      {
      *error_type = 2;
      return 0;
      }
}
```

```
/* Program 4_4 */

/* Delete blank spaces at the end of a string */

#include <stdio.h>

int trim(char s[]);
int length_of_string(char buf[]);

/* main */

void main()
{
   char string1[] = "This is a Test            ";

   printf("New String has length %d\n", trim(string1));
   printf("%s!!!Here is the end of the string\n",string1);
}

/* definition of function length_of_string, see Program 2_11 */

int length_of_string(char buf[])
{
   int l;

   l=0;
   while (buf[l] != '\0')
      l++;

   return l;
}

/* definition of function trim (delete blank spaces) */

int trim(char s[])
{
   int n;

   for (n = length_of_string(s)-1; n>=0; n--)
      if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
         break;
   s[n+1] = '\0';
   return n;
}
```

6 Pointers and Vectors

```
/* Program 6_1 */


#include <stdio.h>


/* main */

void main()
{
   int a,b;
   int *pa, *p;

   a = 6;

   pa = &a;       /* pa points to a */

   b = *pa;       /* pa has now the value 6 */

   (*pa)++;       /* a is now 7      */

   p = &b;        /* p points to b   */

   *p = *pa;      /* b is now 7 */

   p = pa;        /* p points now to a   */

   (*p)++;        /* a is 8  */

   p = &b;        /* p points to b again */


   printf("%d %d %d %d\n",a,b,*pa, *p);

   printf("%p %p\n", pa,p);
}
```

```
/* Program 6_2 */


#include <stdio.h>


/* main */

void main()
{
   int a;

   int v[] = { 0, 10, 20, 30, 40, 50 };

   int *p, *pa;

   p = &v[0];

   a = *p; /* a is 0 */

   printf("%d\n",a);

   pa = &a;

   p++;

   *pa = *p;

   printf("%d\n",a);

}
```

```
/* Program 6_3 */


#include <stdio.h>
#include <math.h>

void polar_to_cartesian(double *a, double *b, double rho, double theta);

void cartesian_to_polar(double a, double b, double *rho, double *theta);


/* main */

void main()
{

   double x,y;
   double r, phi;

   scanf("%lf %lf", &x, &y);
   cartesian_to_polar(x,y,&r, &phi);
   printf("%lf %lf\n", r, phi);

   scanf("%lf %lf", &r, &phi);
   polar_to_cartesian(&x, &y, r,phi);
   printf("%lf %lf\n", x,y);
}



void polar_to_cartesian(double *a, double *b, double rho, double theta)
   {
   *a = rho * cos(theta);
   *b = rho * sin(theta);
   }


void cartesian_to_polar(double a, double b, double *rho, double *theta)
   {
   *rho = sqrt(a*a+b*b);
   *theta = atan2(b/ *rho, a/ *rho);
   }
```

```
/* Program 6_4 */

/* 4 versions of string copy */


#include <stdio.h>


void strcpy1(char *s, char *t);
void strcpy2(char *s, char *t);
void strcpy3(char *s, char *t);
void strcpy4(char *s, char *t);



/* main */

void main()
{
   char string1[] = "This is a Test", string2[80];


   strcpy1(string2, string1);
/* or: */
   strcpy2(string2, string1);
/* or: */
   strcpy3(string2, string1);
/* or: */
   strcpy4(string2, string1);
/* or: */


   printf("%s %s\n",string1,string2);
}

void strcpy1(char *s, char *t)
{
   int i;

   i = 0;
   while ((s[i] = t[i]) != '\0')
      i++;
}

void strcpy2(char *s, char *t)
{
   while ((*s = *t) != '\0') {
      s++;
      t++;
      }
}
```

```
void strcpy3(char *s, char *t)
{
   while ((*s++ = *t++) != '\0')
      ;
}

void strcpy4(char *s, char *t)
{
   while ((*s++ = *t++))
      ;
}
```

```
/* Program 6_5 */

/* Euclidean norm of a vector  (2 versions) */

#include <stdio.h>
#include <math.h>

#define MAXDIM 20


int input_vector(double *v);
double norm(int dim, double *v);



/* main */

void main()
{
   double v[MAXDIM];
   double no;
   int dim;

   dim = input_vector(v);

   no = norm(dim, v);

   printf("The norm is: %20.10lf\n", no);

}
```

```
int input_vector(double *v)
{
   int n,i;

   scanf("%d",&n);

   for (i=0; i<n; i++)
      scanf("%lf", v+i);

   return n;
}

double norm(int dim, double *v)
{
   int i;
   double help = 0.;

   for (i=0; i<dim; i++)
      help += v[i]*v[i];

   return sqrt(help);
}

/* Other versions could read: */

int input_vector(double *v)
{
   int n, i;

   scanf("%d",&n);

   for (i=0; i<n; i++)
      scanf("%lf", v++);

   return n;
}

double norm(int dim, double *v)
{
   double help = 0.;

   while (dim--)
      help += *v * *v++;

   return sqrt(help);
}
```

```
/* Program 6_6 */

/* Sorting */

#include <stdio.h>

#define MAXDIM 20


int input_vector(double *v);
void output_vector(int dim, double *v);
void sort(int dim, double *v);
void swap (double *a, double *b);



/* main */

void main()
{
   double v[MAXDIM];
   double no;
   int dim;

   dim = input_vector(v);

   sort(dim, v);

   output_vector(dim, v);

}


int input_vector(double *v)
{
   int n, i;

   scanf("%d",&n);

   for (i=0; i<n; i++)
      scanf("%lf", v++);

   return n;
}

void output_vector(int dim, double *v)
{
   while (dim--)
      printf("%20.16lf\n", *v++);
}
```

```
void swap(double *a, double *b)
{
   double help;

   help = *a;
   *a = *b;
   *b = help;
}



void sort(int dim, double *v)
{
   int i, j;
   int min;

   for (i=0; i< dim-1; i++) {
      min = i;
      for (j=i+1; j < dim; j++)
         if (v[j] < v[min])
            min = j;
         swap(&v[i], &v[min]);
      }
}



/* Another (recursive) formulation is: (use only one version!)*/


void sort(int dim, double *v)
{
   int i, j;
   double *min;

   if (dim <=1) return;

   min = v;
   for (i=1; i < dim; i++)
      if (*(v+i) < *min)
         min = v+i;
   swap(v, min);

   sort(dim-1, v+1);

}
```

```
/* Program 6_7 */

/* The use of malloc and free */

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


double *input_vector(int *dim);
void output_vector(int dim, double *v);



/* main */

void main()
{
   double *v;
   int dim;

   v = input_vector(&dim);
   if (!v) goto cleanup;

   output_vector(dim, v);


cleanup:

   if (v) free(v);

}

double *input_vector(int *dim)
{
   double *v;
   int i;
   scanf("%d",dim);

   if (dim <= 0) return NULL;

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

   if (!v) return NULL;


   for (i=0; i<*dim; i++)
      scanf("%lf", v+i);

   return v;
}
```

```
void output_vector(int dim, double *v)
{
   while (dim--)
      printf("%20.16lf\n", *v++);
}
```

```
/* Program 6_8 */

/* The same as Program 6_7 with another variable list for the input function */

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


int input_vector(double **);
void output_vector(int dim, double *v);



/* main */

void main()
{
   double *v = NULL;
   int dim;

   dim =  input_vector(&v);
   if (!v) goto cleanup;

   output_vector(dim, v);


cleanup:

   if (v) free(v);

}
```

```
int input_vector(double **v)
{
   int dim, i;
   scanf("%d",&dim);

   if (dim <= 0) return dim;

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

   if (!*v) return dim;

   for (i=0; i<dim; i++)
      scanf("%lf", *v+i);

   return dim;
}
```

```
void output_vector(int dim, double *v)
{
   while (dim--)
      printf("%20.16lf\n", *v++);
}
```

```
/* Program 6_9 */

/* How to store matrices of variable size */

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

double *create_matrix(int rows, int columns);
double *input_matrix(int *rows, int *columns);
void output_matrix(int rows, int columns, double *A);
double *transpose_matrix(int *Brows, int *Bcolumns,
                         int  Arows, int  Acolumns, double *A);


/* main */

void main()
{
   double *A, *B=NULL;
   int rows_A, columns_A;
   int rows_B, columns_B;

   A = input_matrix(&rows_A, &columns_A);
   if (!A) goto cleanup;

   output_matrix(rows_A, columns_A, A);

   B = transpose_matrix(&rows_B, &columns_B, rows_A, columns_A, A);

   if (B) output_matrix(rows_B, columns_B, B);


cleanup:

   if (A) free(A);
   if (B) free(B);

}


double *create_matrix(int rows, int columns)
{
   double *A;

   if (!(A= (double *)malloc(rows*columns*sizeof(double))))
      printf("Allocation failed!\n");
   return A;
}
```

```
double *input_matrix(int *rows, int *columns)
{
   int i,j;
   double *A;
   scanf("%d %d ",rows, columns);

   if (*rows <=0 || *columns <=0) return NULL;

   A = create_matrix(*rows, *columns);

   if (!A) return NULL;

   for (i=0; i< *rows; i++)
      for (j=0; j< *columns; j++)
         scanf("%lf", A+i * *columns+j);

   return A;
}



void output_matrix(int rows, int columns, double *A)
{
   int i;

   for (i=0; i < rows*columns; i++)
      printf("%10.4lf%c", *(A+i), (i % columns == columns-1) ? '\n' : ' ');
}


double *transpose_matrix(int *Brows, int *Bcolumns,
                    int  Arows, int  Acolumns, double *A)
{
   double *B;

   B = create_matrix(Acolumns, Arows);
   if (!B) return NULL;

   for (*Brows = 0; *Brows < Acolumns; (*Brows)++)
      for (*Bcolumns=0; *Bcolumns < Arows; (*Bcolumns)++)
         *(B+ *Brows * Arows + *Bcolumns) =
                    *(A + *Bcolumns * Acolumns + *Brows);

   return B;
}
```

```
/* Program 6_10 */

/* Matrices with a vector of pointer, here with variable number of columns */
/* We are dealing with lower triangular matrices */

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

double **create_triangular_matrix(int dim);
void free_triangular_matrix(int dim, double **A);
double **input_triangular_matrix(int *dim);
void output_triangular_matrix(int dim, double **A);

/* main */

void main()
{
   double **A;
   int dim;

   A = input_triangular_matrix(&dim);
   if (!A) goto cleanup;

   output_triangular_matrix(dim, A);

cleanup:

   free_triangular_matrix(dim, A);
}

double **create_triangular_matrix(int dim)
{
   double **A;
   int i, ok = 1;

   A= (double **)malloc(dim*sizeof(double *));
   if (!A) return NULL;
   for (i=0; i<dim; i++)
      A[i] = NULL;

   for (i=0; i<dim && ok; i++)
      ok = (A[i] = (double *) malloc((i+1)*sizeof(double))) != NULL;

   if (ok) return A;
   else
      {
      free_triangular_matrix(dim, A);
      return NULL;
      }
}

void free_triangular_matrix(int dim, double **A)
```

```
{
   int i;
   if (A)
      {
      for (i=0; i<dim; i++)
         if (A[i]) free(A[i]);
      free(A);
      }
}


double **input_triangular_matrix(int *dim)
{
   int i,j;
   double **A;
   scanf("%d", dim);

   if (*dim <=0) return NULL;

   A = create_triangular_matrix(*dim);

   if (!A) return NULL;

   for (i=0; i< *dim; i++)
      for (j=0; j<=i; j++)
         scanf("%lf", *(A+i) + j);

   return A;
}



void output_triangular_matrix(int dim, double **A)
{
   int i, j;

   for (i=0; i < dim; i++)
      {
      for(j=0; j<=i; j++)
         printf("%10.4lf ", A[i][j]);
      printf("\n");
      }
}
```

```
/* Program 6_11 */

/* An implementation of echo */

#include <stdio.h>


/* main */

void main(int argc, char *argv[])
{
   int i;

   for (i=1; i< argc; i++)
      printf("%s ", argv[i]);
   printf("\n");
}
```

```
/* Program 6_12 */

/* Another implementation of echo */

#include <stdio.h>


/* main */

void main(int argc, char *argv[])
{

   while (--argc >0)
      printf("%s ", *++argv);
   printf("\n");
}
```

```
/* Program 6_13 */

/* Pointer to functions. A simple plot program */

#include <stdio.h>
#include <math.h>

#define WIDTH  60
#define HEIGHT 40
#define PLOT_SYMBOL '*'

/* declaration of functions */

double sample_f(double x);
double sample_g(double x);

void range(double x_min, double x_max, double (*f)(double),
           double *y_min, double *y_max);
void plot(double x_min, double x_max, double (*f)(double));

/* main */

void main()
{
   double (*f)(double);

   f = sample_f;
   plot(0, 2, f);

   plot(0, 6.29, sample_g);

   plot(0, 6.29, sin);
}

void range(double x_min, double x_max, double (*f)(double),
           double *y_min, double *y_max)
{
   int i;
   double y;
   double dx=(x_max - x_min) / (double)HEIGHT;

   *y_min= *y_max = (*f)(x_min);

   for (i=0, x_min += dx; i<HEIGHT; i++, x_min += dx) {
         y = (*f)(x_min);
      if (y > *y_max)
         *y_max = y;
      if (y < *y_min)
         *y_min = y;
      }
}
```

```
void plot(double x_min, double x_max, double (*f)(double))
{
   int i, j, t;
   double x, y;
   double dx = (x_max - x_min) / (double)HEIGHT;
   double y_min, y_max, dy;

   range(x_min, x_max, f, &y_min, &y_max);

   dy = (double) WIDTH / (y_max - y_min);

   for (i=0, x = x_min; i<= HEIGHT; i++, x += dx) {
      y = (*f)(x);
      t = (int)( (y - y_min) * dy + 0.5);
      for (j=0; j<t; j++)
         printf(" ");
      printf("%c\n", PLOT_SYMBOL);
      }
   printf("\n\n\n");
}



double sample_f(double x)
{
   return x*x;
}

double sample_g(double x)
{
   return sin(x)*cos(x);
}
```

```
/* Program 6_14 */

/* A possibility for the efficient storage of sparse matrices.
 * Here: MRS-format (modified compressed row storage)
 */

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

#define abs(x) ((x) > 0 ? (x) : -(x))
#define ZERO 1e-20


/* We use the following functions from Program 6_7 */

double *input_vector(int *dim);
void output_vector(int dim, double *v);


/* New in this program are */


int  input_sparse_matrix(int *dim, float **A, unsigned short **I);
void sparse_matrix_vector_mult(int dim, float *A, unsigned short *I,
                               double *x, double *y);



/* main */

void main()
{
   float *A;
   unsigned short *I;

   double *x, *y;

   int dim, vector_dim;


   if (!input_sparse_matrix(&dim, &A, &I))
      goto cleanup;

   x = input_vector(&vector_dim);
   if (!x) goto cleanup;
   if (dim != vector_dim) {
      printf("Wrong dimension!\n");
      goto cleanup;
      }
```

```
    if (!(y = (double *)malloc(dim*sizeof(double)))) goto cleanup;
    sparse_matrix_vector_mult(dim, A, I, x, y);
    output_vector(dim, y);


cleanup:

    if (A) free(A);
    if (I) free(I);
    if (x) free(x);
    if (y) free(y);

}



double *input_vector(int *dim)
{
    double *v;
    int i;
    scanf("%d",dim);

    if (dim <= 0) return NULL;

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

    if (!v) return NULL;


    for (i=0; i<*dim; i++)
       scanf("%lf", v+i);

    return v;
}


void output_vector(int dim, double *v)
{
    while (dim--)
       printf("%20.16lf\n", *v++);
}
```

```
int  input_sparse_matrix(int *dim, float **A, unsigned short **I)
{
   int size;
   int i,j;
   int next;
   float  *M;
   unsigned short *Index;

   float help;


   printf("Dimension of the matrix:");
   scanf("%d", dim);
   printf("Number of non-zero elements off the diagonal:");
   scanf("%d", &size);

   /* The required number of entries is */

   size += *dim+1;

/* Remark: It is not so nice, that the user has to calculate the
 * required size of M and Index by himself. In an exercise, this
 * will be done automatically
 */


   if (!(M = (float *)malloc(size*sizeof(float))))
      return 0;
   if (!(Index = (unsigned short *)malloc(size*sizeof(unsigned short)))) {
      free(M);
      return 0;
      }

   next = *dim+1;
   for (i=0; i<*dim; i++) {
      Index[i] = next;
      for (j=0; j<*dim; j++) {
         scanf("%f", &help);
         if (i==j) {   /* diagonal elements are stored extra */
             M[i] = help;
             }
         else {
            if (abs(help) > ZERO) {   /* Only in this case,
                                       * the element is stored */
               M[next] = help;
               Index[next] = j;
               next++;
               }
            }
         }
```

```
    }
  Index[*dim] = next; /* next is now equal to size */

  *A = M;
  *I = Index;

  return 1;
}
```

```
void sparse_matrix_vector_mult(int dim, float *A, unsigned short *I,
      double *x, double *y)
{

  int i,j;

  for (i=0; i< dim; i++) {
    y[i] = A[i]*x[i];
    for (j= I[i]; j < I[i+1]; j++)
      y[i] += A[j]*x[I[j]];
    }
}
```

---

7 Structures

```
/* Program 7_1 */

/* Nonsense with structures */

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



void main()
{
   struct point {
      double x;
      double y;
      };

   struct point p1, p2;

   struct point *pp, *pq;
   double x,y;


   p1.x = 17.4;
   p1.y = 3.5;
   x = p1.x;
   y = p1.y;

   p2 = p1;

   pp = &p1;

   x = pp->x;
   pp->y = 11.1;

   pq = (struct point *)malloc(sizeof(struct point));
   if (pq == NULL) exit(1);

   *pq = p1;

   if (pq) free(pq);


}
```

```
/* Program 7_2 */

/* Calculus with fractions */

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


struct fraction {
   long int numerator;
   long int denominator;
   };

/* declaration of functions */

struct fraction input(void);
void output(struct fraction x);
struct fraction  add(struct fraction x, struct fraction y);
struct fraction mult(struct fraction x, struct fraction y);

void main()
{
   struct fraction x,y,z;

   x = input();
   y = input();
   z = add(x,y);

   output(z);
   output(mult(x,y));
}

struct fraction input()
{
   struct fraction x;

   printf("Numerator:");
   scanf("%ld", &x.numerator);
   do {
      printf("Denominator:");
      scanf("%ld", &x.denominator);
      } while (x.denominator == 0);

/* We require that the denominator is not negative ... */

   if (x.denominator <0) {
      x.denominator = - x.denominator;
      x.numerator  = - x.numerator;
      }
   return x;
}
```

```
void output(struct fraction x)
{
   printf("%ld/%ld\n", x.numerator, x.denominator);
}

struct fraction add(struct fraction x, struct fraction y)
{
   struct fraction z;

   z.denominator = x.denominator * y.denominator;
   z.numerator = x.numerator*y.denominator + y.numerator*x.denominator;
   return z;
}

struct fraction mult(struct fraction x, struct fraction y)
{
   x.denominator *= y.denominator;
   x.numerator  *= y.numerator;
   return x;
}
```

```
/* Program 7_3 */

/* Another matrix implementation */

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


struct matrix {
   int rows;
   int columns;
   double *data;
   };

/* declaration of functions */

struct matrix input(void);
void output(struct matrix A);
struct matrix destroy(struct matrix A);
```

```
void main()
{
   struct matrix A;

   A = input();
   if (A.data)
      output(A);
   A = destroy(A);

}

struct matrix input()
{
   struct matrix A;
   int i,j;

   scanf("%d %d", &A.rows, &A.columns);

   A.data = (double *)malloc(A.rows*A.columns*sizeof(double));
   if (A.data) {
      for (i=0; i<A.rows; i++)
         for (j=0;j<A.columns; j++)
            scanf("%lf", A.data+i*A.columns+j);
      }
   return A;
}

void output(struct matrix A)
{
   int i,j;
   if (!A.data) return;
   for (i=0; i<A.rows; i++) {
      for (j=0; j<A.columns; j++)
         printf("%12.8lf ", *(A.data+i*A.columns+j));
      printf("\n");
      }
}

struct matrix destroy(struct matrix A)
{
   if (A.data) {
      free(A.data);
      A.data = NULL;
      }
   return A;
}
```

```
/* Program 7_4 */

/* The matrices of Program 7_3, this time with pointers */

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


struct matrix {
   int rows;
   int columns;
   double *data;
   };

/* declaration of functions */

struct matrix *input(void);
void output(struct matrix *A);
void destroy(struct matrix **A);

void main()
{
   struct matrix *A;

   A = input();
   output(A);
   destroy(&A);

}

struct matrix *input()
{
   struct matrix *A;
   int i,j;

   if (!(A = (struct matrix *)malloc(sizeof(struct matrix))))
      return NULL;
   scanf("%d %d", &A->rows, &A->columns);

   A->data = (double *)malloc(A->rows * A->columns * sizeof(double));
   if (!A->data) {
      free(A);
      return NULL;
      }
   for (i=0; i<A->rows; i++)
      for (j=0;j<A->columns; j++)
         scanf("%lf", A->data+i * A->columns + j);
   return A;
}
```

```
void output(struct matrix *A)
{
   int i,j;
   if (!A->data) return;
   for (i=0; i<A->rows; i++) {
      for (j=0; j<A->columns; j++)
         printf("%12.8lf ", *(A->data+i * A->columns + j));
      printf("\n");
      }
}

void destroy(struct matrix **A)
{
   if (*A) {
      if ((*A)->data)
         free((*A)->data);
      free (*A);
      *A = NULL;
      }
}
```

```
/* Program 7_5 */

/* Chained list */

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


/* The following lines would normally be contained
 * in a header file.
 */

typedef struct person *ptrPerson;
typedef struct person {
   int number;
   char name[40];
   ptrPerson next;
   } Person;

/* declaration of functions */

ptrPerson lookup(int number, ptrPerson p);
void input(ptrPerson p);
void output(ptrPerson p);
void kill(ptrPerson p);
void clear(ptrPerson p);
void do_the_loop(ptrPerson p);


void main()
{
   Person start = { 0, "Starting node", NULL};

   do_the_loop(&start);

   clear(&start);
}

void input(ptrPerson p)
{
   int number;
   ptrPerson h, x;

   printf("Number:");
   scanf("%d", &number);

   if (h=lookup(number, p)) {
      printf("Person already in list.\n");
      printf("The name is: %s\n", h->next->name);
      return;
      }
```

```
/* Find the right position in the list... */

   x = p;
   while(x->next && x->next->number < number)
      x = x->next;
   h = x->next;
   x->next = (ptrPerson)malloc(sizeof(Person));
   if (!(x->next)) {
      printf("No memory!\n");
      return;
      }
   x->next->number = number;
   x->next->next = h;

   printf("Name:");
   scanf("%s", x->next->name);
}


void output(ptrPerson p)
{
   ptrPerson x;
   x = p;
   while (x->next) {
      x = x->next;
      printf("%3d %s", x->number, x->name);
      }
   printf("\n");
}


void clear(ptrPerson p)
{
   if (p->next) {
      clear(p->next);
      free(p->next);
      }
   p->next = NULL;
}

ptrPerson lookup (int number, ptrPerson p)
{
   while(p->next) {
      if (p->next->number == number)
         return p;
      p = p->next;
      }
   return NULL;
}
```

```
void kill(ptrPerson p)
{
   int number;
   ptrPerson h, x;

   printf("Number of the person to be killed:");
   scanf("%d", &number);

   if (!(x=lookup(number, p))) {
      printf("Person not in list.\n");
      return;
      }
   h = x->next->next;
   free(x->next);
   x->next = h;
}

void do_the_loop(ptrPerson p)
{
   int choice;

   for(;;) {
      printf("Input new person: 1\n");
      printf("Kill person:      2\n");
      printf("Show list:        3\n");
      printf("Delete all:       4\n");
      printf("Leave program:    5\n");
      printf("\n");
      printf("Your choice:      ");
      scanf("%d", &choice);

      switch(choice) {
         case 1:
            input(p);
            break;
         case 2:
            kill(p);
            break;
         case 3:
            output(p);
            break;
         case 4:
            clear(p);
            break;
         case 5:
            return;
            break;
         }
      }
}
```

8 Input and Output

```
/* Program 8_1 */

/* Change upper to lower characters from stdin to stdout
 * or vice versa (dependent on the name of the program)
 */

#include <stdio.h>
#include <string.h>

#define UPPER_NAME "upper"
#define LOWER_NAME "lower"

void do_upper(void);
void do_lower(void);

void main(int argc, char *argv[])
{

if (strcmp(argv[0],UPPER_NAME) == 0)
   do_upper();
else if (strcmp(argv[0], LOWER_NAME) == 0)
   do_lower();
}

void do_upper()
{

int c;

   while ((c=getchar()) != EOF) {
      if (c >= 'a' && c <= 'z')
         c += 'A' - 'a';
      putchar(c);
      }
}

void do_lower()
{
int c;

   while ((c=getchar()) != EOF) {
      if (c >= 'A' && c <= 'Z')
         c += 'a' - 'A';
      putchar(c);
      }
}
```

```
/* Program 8_2 */

/* An easy implementation of cat */

#include <stdio.h>

void filecopy(FILE *fp1, FILE *fp2);

void main(int argc, char *argv[])
{

FILE *fp;

   while (--argc >0)
      if ((fp = fopen(*++argv, "r")) == NULL) {
         printf("Can't open file %s\n", *argv);
         return;
         }
      else {
         filecopy(fp, stdout);
         fclose(fp);
         }
   return;
}

void filecopy(FILE *fp1, FILE *fp2)
{
   int c;

   while ((c = getc(fp1)) != EOF)
      putc(c, fp2);
}
```

```
/* Program 8_3 */

/* How to read and write  matrices from and in a file, version 1 */

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

double *read_matrix(char *filename, int *rows, int *columns);
void write_matrix(char *filename, int rows, int columns, double *A);
void output_matrix(int rows, int columns, double *A);
```

```
void main()
{

int rows, columns;
double *A;

A = read_matrix("test", &rows, &columns);

if (A) {
   printf("Successfully read matrix with %d rows and %d columns.\n",
            rows, columns);
   output_matrix(rows, columns, A);
   write_matrix("test_new", rows, columns, A);
   free(A);
   }
}

double *read_matrix(char *filename, int *rows, int *columns)
{

FILE *fp;
double *A;
register int i,j;

   if ((fp = fopen(filename, "r")) == NULL) {
      printf("Can't open file %s\n", filename);
      return NULL;
      }

   if (fscanf(fp, "%d %d", rows, columns) != 2) {
      printf("Error while reading file %s\n", filename);
      fclose(fp);
      return NULL;
      }

   A = (double *)malloc(*rows * *columns * sizeof(double));
   if (!A) {
      printf("Can't allocate %d Bytes\n", *rows * *columns * sizeof(double));
      fclose(fp);
      return NULL;
      }

   for (i=0; i < *rows; i++)
      for (j=0; j < *columns; j++)
         if (fscanf(fp, "%lf", A+i * *columns + j) != 1) {
            printf("Error while reading file %s\n", filename);
            free(A);
            fclose(fp);
            return NULL;
            }
```

```
    fclose(fp);
    return A;
}



void write_matrix(char *filename, int rows, int columns, double *A)
{

FILE *fp;
register int i,j;

    if ((fp = fopen(filename, "w")) == NULL) {
        printf("Can't open file %s\n", filename);
        return;
        }

    if (fprintf(fp, "%d %d\n", rows, columns) == EOF) {
        printf("Error while writing file %s\n", filename);
        fclose(fp);
        return;
        }

    for (i=0; i < rows; i++)
        for (j=0; j < columns; j++)
            if (fprintf(fp, "%lf %c", *(A+i * columns + j),
                    j+1 == columns ? '\n' : ' ') == EOF) {
                printf("Error while writing file %s\n", filename);
                fclose(fp);
                return;
                }

    fclose(fp);
}

/* The following function is copied from
 * Program 6_9
 */

void output_matrix(int rows, int columns, double *A)
{
    int i;

    for (i=0; i < rows*columns; i++)
        printf("%10.4lf%c", *(A+i), (i % columns == columns-1) ? '\n' : ' ');
}
```

```
/* Program 8_4 */

/* How to read and write  matrices from and in a file, version 2.
 * Binary storage.
 */

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

double *read_matrix(char *filename, int *rows, int *columns);
void write_matrix(char *filename, int rows, int columns, double *A);
double *read_matrix_binary(char *filename, int *rows, int *columns);
void write_matrix_binary(char *filename, int rows, int columns, double *A);
void output_matrix(int rows, int columns, double *A);


void main()
{

int rows, columns;
double *A;

A = read_matrix("test", &rows, &columns);

if (A) {
   printf("Successfully read matrix with %d rows and %d columns.\n",
        rows, columns);
   write_matrix_binary("test_bin", rows, columns, A);

   free(A);

   A = read_matrix_binary("test_bin", &rows, &columns);

   if (A) {
      printf("Successfully read (binary) matrix with %d rows and %d columns.\n",
           rows, columns);
      output_matrix(rows, columns, A);
      free(A);
      }
   }
}

double *read_matrix(char *filename, int *rows, int *columns)
{

FILE *fp;
double *A;
register int i,j;
```

```
    if ((fp = fopen(filename, "r")) == NULL) {
       printf("Can't open file %s\n", filename);
       return NULL;
       }

    if (fscanf(fp, "%d %d", rows, columns) != 2) {
       printf("Error while reading file %s\n", filename);
       return NULL;
       }

    A = (double *)malloc(*rows * *columns * sizeof(double));
    if (!A) {
       printf("Can't allocate %d Bytes\n", *rows * *columns * sizeof(double));
       fclose(fp);
       return NULL;
       }

    for (i=0; i < *rows; i++)
       for (j=0; j < *columns; j++)
          if (fscanf(fp, "%lf", A+i * *columns + j) != 1) {
             printf("Error while reading file %s\n", filename);
             free(A);
             fclose(fp);
             return NULL;
             }

    fclose(fp);
    return A;
}

double *read_matrix_binary(char *filename, int *rows, int *columns)
{

FILE *fp;
double *A;
register int i,j;

    if ((fp = fopen(filename, "r")) == NULL) {
       printf("Can't open file %s\n", filename);
       return NULL;
       }

    if (fread(rows, sizeof(int), 1, fp) != 1) {
       printf("Error while reading file %s\n", filename);
       return NULL;
       }
    if (fread(columns, sizeof(int), 1, fp) != 1) {
       printf("Error while reading file %s\n", filename);
       return NULL;
       }
```

```
    A = (double *)malloc(*rows * *columns * sizeof(double));
    if (!A) {
       printf("Can't allocate %d Bytes\n", *rows * *columns * sizeof(double));
       fclose(fp);
       return NULL;
       }

    if (fread(A, sizeof(double), *rows * *columns, fp) != *rows * *columns) {
       printf("Error while reading file %s\n", filename);
       fclose(fp);
       free(A);
       return NULL;
       }

    fclose(fp);
    return A;
}


void write_matrix(char *filename, int rows, int columns, double *A)
{

FILE *fp;
register int i,j;

    if ((fp = fopen(filename, "w")) == NULL) {
       printf("Can't open file %s\n", filename);
       return;
       }

    if (fprintf(fp, "%d %d\n", rows, columns) == EOF) {
       printf("Error while writing file %s\n", filename);
       fclose(fp);
       return;
       }

    for (i=0; i < rows; i++)
       for (j=0; j < columns; j++)
          if (fprintf(fp, "%lf %c", *(A+i * columns + j),
                j+1 == columns ? '\n' : ' ') == EOF) {
             printf("Error while writing file %s\n", filename);
             fclose(fp);
             return;
             }

    fclose(fp);
}
```

```
void write_matrix_binary(char *filename, int rows, int columns, double *A)
{

FILE *fp;
register int i,j;

   if ((fp = fopen(filename, "w")) == NULL) {
      printf("Can't open file %s\n", filename);
      return;
      }

   if (fwrite(&rows, sizeof(int), 1, fp) != 1) {
      printf("Error while writing file %s\n", filename);
      fclose(fp);
      return;
      }

   if (fwrite(&columns,sizeof(int), 1, fp) != 1) {
      printf("Error while writing file %s\n", filename);
      fclose(fp);
      return;
      }

   if (fwrite(A,sizeof(double), rows * columns, fp) != rows * columns) {
      printf("Error while writing file %s\n", filename);
      fclose(fp);
      return;
      }


   fclose(fp);
}



/* The following function is copied from
 * Program 6_9
 */


void output_matrix(int rows, int columns, double *A)
{
   int i;

   for (i=0; i < rows*columns; i++)
      printf("%10.4lf%c", *(A+i), (i % columns == columns-1) ? '\n' : ' ');
}
```

```
/* Program 8_5 */

/* A better implementation of cat
 * ... including error handling
 */



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

void filecopy(FILE *fp1, FILE *fp2);

void main(int argc, char *argv[])
{

FILE *fp;
char *progname = argv[0]; /* Name of program
                           * for error message */
   while (--argc >0)
      if ((fp = fopen(*++argv, "r")) == NULL) {
         fprintf(stderr, "%s: Can't open file %s\n", progname,  *argv);
         exit(1);
         }
      else {
         filecopy(fp, stdout);
         fclose(fp);
         }
   if (ferror(stdout)) {
      fprintf(stderr, "%s: Error while writing stdout\n", progname);
      exit(2);
      }
   exit(0);
}

void filecopy(FILE *fp1, FILE *fp2)
{
   int c;

   while ((c = getc(fp1)) != EOF)
      putc(c, fp2);
}
```

```
/* Program 8_6 */

/* Chained list with read and write (extension of Program 7_5) */


/* The following lines would normally contained
 * in a header file.
 */

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

typedef struct person *ptrPerson;
typedef struct person {
   int number;
   char name[40];
   ptrPerson next;
   } Person;

/* declaration of functions */

ptrPerson lookup(int number, ptrPerson p);
void input(ptrPerson p);
void output(ptrPerson p);
void kill(ptrPerson p);
void clear(ptrPerson p);
void do_the_loop(ptrPerson p);
void read_from_file(ptrPerson p);
void write_to_file(ptrPerson p);

void main()
{
   Person start = { 0, "Starting node", NULL};
   do_the_loop(&start);
   clear(&start);
}

void input(ptrPerson p)
{
   int number;
   ptrPerson h, x;

   printf("Number:");
   scanf("%d", &number);

   if (h=lookup(number, p)) {
      printf("Person already in list.\n");
      printf("The name is: %s\n", h->next->name);
      return;
      }
```

```
/* Find the right position in the list... */

   x = p;
   while(x->next && x->next->number < number)
      x = x->next;
   h = x->next;
   x->next = (ptrPerson)malloc(sizeof(Person));
   if (!(x->next)) {
      printf("No memory!\n");
      return;
      }
   x->next->number = number;
   x->next->next = h;

   printf("Name:");
   scanf("%s", x->next->name);
}


void output(ptrPerson p)
{
   ptrPerson x;
   x = p;
   while (x->next) {
      x = x->next;
      printf("%3d %s", x->number, x->name);
      }
   printf("\n");
}


void clear(ptrPerson p)
{
   if (p->next) {
      clear(p->next);
      free(p->next);
      }
   p->next = NULL;
}

ptrPerson lookup(int number, ptrPerson p)
{
   while(p->next) {
      if (p->next->number == number)
         return p;
      p = p->next;
      }
   return NULL;
}
```

```
void kill(ptrPerson p)
{
   int number;
   ptrPerson h, x;

   printf("Number of the person to be killed:");
   scanf("%d", &number);

   if (!(x=lookup(number, p))) {
      printf("Person not in list.\n");
      return;
      }
   h = x->next->next;
   free(x->next);
   x->next = h;

}


void write_to_file(ptrPerson p)
{
   FILE *fp;
   char filename[80];
   ptrPerson x=p;

   printf("Input filename: ");
   scanf("%s",filename);

   if (!(fp = fopen(filename, "w"))) {
      printf("Kann file %s nicht oeffnen.\n", filename);
      return;
      }

   while (x->next) {
      x = x->next;
      if (fprintf(fp,"%d\n%s\n", x->number, x->name) == EOF) {
         printf("Fehler beim Schreiben des files %s\n", filename);
         fclose(fp);
         return;
         }
      }
   if (fprintf(fp, "%d\n",0) == EOF) {
      printf("Fehler beim Schreiben des files %s\n", filename);
      fclose(fp);
      return;
      }
  fclose(fp);
}
```

```
void read_from_file(ptrPerson p)
{
   FILE *fp;
   char filename[80];
   char name[40];
   int number;
   int weiter = 1, error = 0;
   ptrPerson x=p;

   clear(p);
   printf("Input filename: ");
   scanf("%s", filename);

   if (!(fp = fopen(filename, "r"))) {
      printf("Kann file %s nicht oeffnen.\n", filename);
      return;
      }

   while (weiter) {
      if (fscanf(fp, "%d %s", &number, name) == EOF) {
         error = 1;
         weiter = 0;
         break;
         }
      if (number == 0) {
         weiter = 0;
         break;
         }
      x->next = (ptrPerson)malloc(sizeof(Person));
      if (!(x->next)) {
         error = 2;
         weiter = 0;
         break;
         }
      x = x->next;
      x->next = NULL;
      x->number = number;
      strcpy(x->name, name);
      }
   if (error) {
      switch(error) {
         case 1:
            printf("Fehler beim Lesen von File %s\n", filename);
            break;
         case 2:
            printf("Fehler beim Allokieren\n");
            break;
         }
      clear(p);
      }
   return;
}
```

```
void do_the_loop(ptrPerson p)
{
   int choice;

   for(;;) {
      printf("Input New Person: 1\n");
      printf("Kill Person:      2\n");
      printf("Show List:        3\n");
      printf("Delete All        4\n");
      printf("Read from file    5\n");
      printf("Write to file     6\n");
      printf("Leave program     7\n");
      printf("\n");
      printf("Your choice:      ");
      scanf("%d", &choice);

      switch(choice) {
         case 1:
            input(p);
            break;
         case 2:
            kill(p);
            break;
         case 3:
            output(p);
            break;
         case 4:
            clear(p);
            break;
         case 5:
            read_from_file(p);
            break;
         case 6:
            write_to_file(p);
            break;
         case 7:
            return;
            break;
      }
   }
}
```