```c
#define PROGNAME "ED_tridiag"
#define VERSION "0.3"
#define DATE  "27.06.2007"
#define AUTHOR "Nils Bluemer"

/* performs QR iterations of a tridiagonal matrix
    NEW (0.2) global shift */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "pointer_utils.c"

void error(char error_text[])
/* standard error handler */
{
  fprintf(stderr,"\n %s run-time error\n", PROGNAME);
    fprintf(stderr,"--%s--\n",error_text);
    fprintf(stderr,"for general help use option -h\n");
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

void printhelp ()
{
printf("************************************************************\n");
printf("%s:  eigenvalue utility\n",PROGNAME);
printf("Version:  %s, %s by %s\n",VERSION,DATE,AUTHOR);
printf("Input:  N lines d[n],e[n], preceded by dimension N,\n");
printf("e.g.:  3\n      1 0\n      2 1\n      3 1\n");
printf("options:  -d# digits on output\n");
printf("         -v# verbosity (default:1)\n");
printf("         -f only print full matrix (no ED)\n");
printf("         -a adjust shift\n");
printf("         -h this help\n");
}

void read_matrix(double *d, double *e, int size)
{
  int i;
  double tmpd,tmpe;

  for (i=1;i<=size;i++){
    scanf("%lf %lf", &tmpd, &tmpe);
      d[i]=tmpd;
      e[i]=tmpe;
    }
  e[1]=0.0;
}

void print_matrix(double *d, double *e, int size, int digits, double shift)
{
  int i;
```

```
  char buffer [20];

  sprintf(buffer,"%%%d.%dlf %%%d.%dlf\n",digits+4,digits,digits+4,digits);

  printf ("========================\n");
  for (i=1;i<=size;i++)
    printf(buffer,d[i]+shift,e[i]);
  printf ("========================\n");
}

void print_full_matrix(double *d, double *e, int size, int digits)
{
  char buffer [20];
  int i,j;

  sprintf(buffer,"%%%d.%dlf  ",digits+4,digits);

  printf("%d\n",size);
  for (i=1;i<=size;i++){
    for (j=1;j<=size;j++)
      if (i==j)
printf(buffer,d[i]);
      else if (i==j+1)
printf(buffer,e[i]);
      else if (i==j-1)
printf(buffer,e[j]);
      else
printf(buffer,0.0);
    printf("\n");
  }
}

double lower_bound(double *d, double *e, int size)
{
  int i;
  double x,bound;

  bound=d[1]-fabs(e[2]);
  x=d[size]-fabs(e[size]);
  if (x<bound)
    bound=x;
  for (i=2;i<size;i++){
    x=d[i]-fabs(e[i])-fabs(e[i+1]);
    if (x<bound)
      bound=x;
  }
  return(bound);
}

/* void jacobi(double **a, int n, double d[], double **v, int *nrot, int *sweeps, int verbosity, int digits) */

void tridiag_qr(double *d, double *e, int size, int *nrot, int *sweeps, int verbosity, int
digits, int maxits)
{
```

```
  int it, n;
  double t, sq, cq, sc, c, s, dn, dnt, ent, diff, x;


  for (it=1;it<=maxits;it++){
    (*sweeps)++;
    dnt=d[1]; /* diagonal element after half similarity trafo */
    ent=e[2];
    x=0;       /* element below tridiagonal */

    for (n=1;n<=size-1;n++){
      nrot+=1;
      if (dnt!=0){
t=-ent/dnt;
cq=1.0/(1+t*t); /* c^2 */
sc=t*cq;        /* s*c */
sq=t*sc;        /* s^2 */
c=sqrt(cq);
s=t*c;
      } else {
sq=1.0;
cq=0.0;
sc=0.0;
c=0.0;
s=1.0;
      }
      diff = 2*sc*e[n+1];
      dn = d[n];
      dnt = s*e[n+1]+c*d[n+1];
/*        printf ("n=%d, s=%lf, c=%lf, dn=%lf, dnt=%lf, x=%lf\n",n, s, c, dn,dnt, x); */

      if (n>1)
e[n]=c*e[n]-s*x;
      if (n<size-1){
x = -s*e[n+2];
ent = e[n+2];
e[n+2] = c*e[n+2];
      }
      e[n+1] = (1.0-2*sq)*e[n+1] + sc*(d[n]-d[n+1]);
      d[n]  = cq*dn + sq*d[n+1] - diff;
      d[n+1] = sq*dn + cq*d[n+1] + diff;
      if (verbosity>1)
print_matrix(d,e,size,digits, 0);
    }
  }
}


int main (int argc, char *argv[])
{
  char c;
  int i, full, imax, adjust_shift;
  int digits, size, nrot, sweeps, verbosity, corr;
  double *d, *e, x, y, z, shift, shiftc, max;
```

```
  digits=5;
  verbosity=0;
  corr=0;
  full=0;
  adjust_shift=0;

  while (--argc > 0 && (*++argv)[0] == '-')
    while (c= *++argv[0])
    switch (c) {
    case 'd':
       sscanf(++argv[0],"%d\n",&digits);
             break;
    case 'v':
       sscanf(++argv[0],"%d\n",&verbosity);
             break;
    case 'f':
       full=1;
             break;
    case 'a':
       adjust_shift=1;
             break;
    case 'h':
      printhelp();
      exit(0);
/*       default:   */
/*         error("No valid choice");   */
    }

  scanf("%d",&size);
  d=dvector(1,size);
  e=dvector(1,size);

  read_matrix(d,e,size);

  if (full>0)
    print_full_matrix(d,e,size,digits);
  else {
    if (verbosity>0){
      printf("Original matrix A:\n");
      print_matrix(d,e,size,digits,0);
    }

    shift=lower_bound(d,e,size);
    if (verbosity>0)
      printf("# shift=%lf\n",shift);

    for (i=1;i<=size;i++)
      d[i]=d[i]-shift;

    nrot=0; sweeps=0;

    do {
```

```
      tridiag_qr(d,e,size,&nrot,&sweeps,verbosity,digits, 10);

      if (verbosity>0){
printf("Transformed matrix (after %d iterations, shift=%lf):\n", sweeps, shift);
print_matrix(d,e,size,digits, shift);
      }
      x=fabs(d[1]);
      y=0;
      max=0;
      for (i=2;i<=size;i++){
x+=fabs(d[i]);
z=fabs(e[i]);
y+=z;
if (z>max){
   max=z;
   imax=i;
}
      }
      if (adjust_shift>0){
  shiftc=d[imax];
shift+=shiftc;
for (i=1;i<=size;i++)
   d[i]-=shiftc;
      }
    } while (y/x>0.00000001);

    printf("== Transformed matrix (after %d iterations):\n", sweeps);
    print_matrix(d,e,size,digits, shift);
  }

  free_dvector(d,1,size);
  free_dvector(e,1,size);
}
```