```
CC= cc -O

HEADERS=global.h
OBJECTS= main.o inout.o physical.o neighbourlist.o solve.o forces.o
TARGET= mdmorse


mdmorse:                $(OBJECTS)
                        $(CC) -o $(TARGET) $(OBJECTS)  -lm


main.o:                 main.c $(HEADERS)
                        $(CC) -c main.c

inout.o:                inout.c $(HEADERS)
                        $(CC) -c inout.c

physical.o:             physical.c $(HEADERS)
                        $(CC) -c physical.c

neighbourlist.o:        neighbourlist.c $(HEADERS)
                        $(CC) -c neighbourlist.c

solve.o:                solve.c $(HEADERS)
                        $(CC) -c solve.c

forces.o:               forces.c $(HEADERS)
                        $(CC) -c forces.c
```

```
//
// Simple MD program made for demonstration purposes
//
//
// Internal units:
//
// All lengths in Å = 1e-10 m
// All times in fs = 1e-15 s
// All velocities and accelerations in Å/fs and Å/fs^2
// All forces in eV/Å
// All energies in eV
//

// Note that since this code is converted from Fortran, array
// indices run from 1 to N, not from 0 to N-1 as in usually in C

#define MAIN
#include "global.h"

main()
{

  // Atom data containing arrays
  //
  // x, v and a are the atom positions, velocities and accelerations
  // Epot and Ekin are the potential and kinetic energy
  // atomname is simply the atom name
  //
  struct vector   x[MAXAT+1],v[MAXAT+1],a[MAXAT+1];
  double   Epot[MAXAT+1],Ekin[MAXAT+1];
  char atomname[MAXAT+1][5];
  int      neighbourlist[MAXAT*MAXNEIGHBOURS+1];

  // Other parameters and variables
  //
  // time is the current time, tmax the maximum time, deltat the time step
  //   and itime the integer number of the current time step
  // T is the Temperature (in K) and m is the atom mass
  // Ekinsum, Epotsum and Etot are the total energies for all atoms
  // N is the number of atoms
  // nneighbourlistupdate determines the interval between neighbour list
  //    updates, and nmovieoutput the interval between atom position output.
  // rpotcut and rskincut are the potential and neighbour list cutoff radii
  // size is the simulation cell size
  // periodic determines whether periodic boundaries will be used (1=yes, 0=no)
  //
  double   time,tmax,deltat,T,Ekinsum,Epotsum,Etot,m;
  int        N,nneighbourlistupdate,itime,nmovieoutput;
  double   rpotcut,rskincut;
  struct vector   size;
  struct ivector  periodic;

  // Morse potential parameters
  double  morseDe,morsealpha,morseRe;

  // Parameters for Berendsen temperature control [J. Chem. Phys.
  // 81 (1984) 3684]
  // btctau is the Berendsen temperature control tau, initialT the initial
  // temperature and desiredT the desired temperature.
  double btctau,initialT,desiredT;
```



```
  // Parameters for Berendsen pressure control [J. Chem. Phys.
  //  81 (1984) 3684]
  // bpctau is the Berendsen pressure control tau, bpcbeta 1/B,
  // P the instantaneous pressure, and desiredP the desired pressure
  double bpctau,bpcbeta,P,desiredP,virial,mu;


  int  i;

  printf("\n--------------- mdmorse03 V1.2 --------------------\n\n");
  //
  // Initialize the simulation
  //
  ReadParams(&T,&m,&size,&tmax,&periodic,
             &deltat,&nneighbourlistupdate,&rpotcut,&rskincut,
             &morseDe,&morsealpha,&morseRe,&nmovieoutput,
             &btctau,&initialT,&bpctau,&bpcbeta,&desiredP);

  // btc: Store desired T for temperature control
  desiredT=T;

  ReadAtoms(x,atomname,&N);

  // Factor 2 because of energy equipartition theorem
  SetTemperature(v,N,m,2.0*initialT);
  // Check that we got it right
  GetTemperature(v,N,m,&T);
  printf("Initial atom temperature is %g\n",T);

  // Initialize accelerations to 0
  for (i=1;i<=N;i++) {
     a[i].x=0.0; a[i].y=0.0; a[i].z=0.0;
  }

  // Start main loop over times
  time=0.0;
  itime=0;
  while (1) {
    if (itime%nneighbourlistupdate==0) {
      UpdateNeighbourlist(x,N,size,periodic,neighbourlist,rskincut);
    }

    Solve1(x,v,a,N,size,periodic,deltat);

    GetForces(x,a,N,size,periodic,m,neighbourlist,rpotcut,
              morseDe,morsealpha,morseRe,Epot,&virial);

    GetTemperature(v,N,m,&T);

    Solve2(v,a,N,deltat,btctau,T,desiredT);

    // Update time
    time=time+deltat;

    // Actual solution done, now compute result quantities

    GetEnergies(v,N,m,&Ekinsum,&Epotsum,&Etot,Ekin,Epot);

    printf("%s %10.3f %10.3f %12.5f %12.5f %12.5f %12.5f\n","ec ",
           time,T,Ekinsum/N,Epotsum/N,Etot/N,P);

    if (itime%nmovieoutput==0) {
```

```
      printf("%s %10.3f\n","Outputting atom movie at t = ",time);
      WriteAtoms(x,atomname,Ekin,Epot,N,time,size);
    }

    // Do Berendsen pressure control
    if (bpctau > 0.0) {
      mu=pow((1.0-bpcbeta*deltat/bpctau*(desiredP-P)),(1.0/3.0));

      size.x=size.x*mu;
      size.y=size.y*mu;
      size.z=size.z*mu;
      for (i=1;i<=N;i++) {
        x[i].x=x[i].x*mu;
        x[i].y=x[i].y*mu;
        x[i].z=x[i].z*mu;
      }
      if (itime%10==1) {
        printf("bpc %10.6f %10.6f %10.6f %12.5f %12.5f %12.5f\n",
               size.x,size.y,size.z,size.x*size.y*size.z,P,mu);
      }
    }


    // Check whether we are done
    if (time >= tmax) break;

    itime=itime+1;

  } // End of loop over time steps

  // Always output final energy and atom coordinates
  // using high precision
  printf("%s %12.5f %12.5f %14.7f %14.7f %14.7f\n","ec ",
         time,T,Ekinsum/N,Epotsum/N,Etot/N);

  WriteAtoms(x,atomname,Ekin,Epot,N,time,size);

}
```

```c
#ifdef MAIN
#define EXTERN
#else
#define EXTERN extern
#endif


EXTERN int seed;

// Define a vector type
struct vector {
  double  x,y,z;
};

// Define an int vector type
struct ivector {
  int  x,y,z;
};

#define  pi 3.14159265358979
#define  e 1.6021892e-19
#define  u 1.660565e-27
#define  kB 1.380662e-23

// Define some needed unit conversion factors
//
// To get internal units from SI, divide by one of these
// To get SI from internal units, multiply by one of these
//
#define  lunit 1e-10
#define  tunit 1e-15
#define  vunit (lunit/tunit)
#define  aunit (vunit/tunit)

// Set array sizes

#define  MAXAT 10000
#define  MAXNEIGHBOURS 100

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

```
// Input-output subroutines for the mdmorse code

#include "global.h"

void ReadParams(initialT,m,size,tmax,periodic,
                deltat,nneighbourlistupdate,rpotcut,rskincut,
                morseDe,morsealpha,morseRe,nmovieoutput,
                btctau,desiredT,bpctau,bpcbeta,desiredP)

double   *initialT,*m,*tmax;
struct vector   *size;
struct ivector  *periodic;
double   *deltat,*rpotcut,*rskincut,*morseDe,*morsealpha,*morseRe;
int      *nneighbourlistupdate,*nmovieoutput;
double   *btctau,*desiredT,*bpctau,*bpcbeta,*desiredP;

{
  int  i,n;
  FILE *fp;
  char  string[40],buf[120],*c;
  double x;

  fp=fopen("mdmorse.in","r");

  if (fp==NULL) { puts("Open error for file mdmorse.in"); exit(1); }

  // Initialize variables to sensible (?) default values
  *initialT=0.0; *m=63.546;
  size->x=0.0; size->y=0.0; size->z=0.0;

  *deltat=1.0;
  *nneighbourlistupdate=5;
  *rpotcut=4.02; *rskincut=5.0;
  // Initialize Morse parameters for Cu
  // Values from Girifalco and Weizer, Phys. Rev. 114 (1959) 687.
  *morseDe=0.3429; *morsealpha=1.3588; *morseRe=2.866;
  *btctau=0.0; *desiredT=0.0;
  *bpctau=0.0; *bpcbeta=1e-3; *desiredP=0.0;

  seed=23267761;



  // Loop through file and find all variables
  i=0;
  while(1) {
     i=i+1;
     c=fgets(buf,120,fp);
     if (c==NULL) break;

     string[0]=' ';
     n=sscanf(buf,"%s %lg",string,&x);

     // If not parameter line, skip this line
     if (string[0] != '-') continue;

     if (n<2) {
       printf("ERROR: Line %-40s looks like data line\n",buf);
       printf("but does not contain data in correct format\n");
       exit(0);
     }
```



```
     // Look for variable-defining strings
     if (strcmp(string,"-initialT")==0) *initialT=x;
     else if (strcmp(string,"-desiredT")==0) *desiredT=x;
     else if (strcmp(string,"-btctau")==0) *btctau=x;

     else if (strcmp(string,"-mass")==0) *m=x;

     else if (strcmp(string,"-bpctau")==0) *bpctau=x;
     else if (strcmp(string,"-bpcbeta")==0) *bpcbeta=x;
     else if (strcmp(string,"-desiredP")==0) *desiredP=x;

     else if (strcmp(string,"-xsize")==0) size->x=x;
     else if (strcmp(string,"-ysize")==0) size->y=x;
     else if (strcmp(string,"-zsize")==0) size->z=x;

     else if (strcmp(string,"-periodicx")==0) periodic->x=(int)(x+0.5);
     else if (strcmp(string,"-periodicy")==0) periodic->y=(int)(x+0.5);
     else if (strcmp(string,"-periodicz")==0) periodic->z=(int)(x+0.5);

     else if (strcmp(string,"-seed")==0) seed=(int)(x+0.5);

     else if (strcmp(string,"-tmax")==0) *tmax=x;
     else if (strcmp(string,"-deltat")==0) *deltat=x;

     else if (strcmp(string,"-nupdate")==0) *nneighbourlistupdate=(int)(x+0.5);

     else if (strcmp(string,"-rpotcut")==0) *rpotcut=x;
     else if (strcmp(string,"-rskincut")==0) *rskincut=x;

     else if (strcmp(string,"-morseDe")==0) *morseDe=x;
     else if (strcmp(string,"-morsealpha")==0) *morsealpha=x;
     else if (strcmp(string,"-morseRe")==0) *morseRe=x;

     else if (strcmp(string,"-nmovieoutput")==0) *nmovieoutput=(int)(x+0.5);

     else {
       printf("Unknown parameter %s in input file\n",string);
       exit(0);
     }

     printf("Read in parameter %s to value %g\n",string,x);

  } // End of readin loop

  fclose(fp);

  // Print out some general information about read results:

  printf("%s %3d %3d %3d\n","Using periodics (1=on, 0=off)",
         periodic->x,periodic->y,periodic->z);

  printf("%s %12.6g %12.6g %12.6g\n","Morse potential parameters: De alpha Re",
       *morseDe,*morsealpha,*morseRe);

  printf("%s %d %s\n","Movie output selected every",*nmovieoutput," steps");

  if (*btctau > 0.0) {
    printf("Doing Berendsen temperature control with tau T %g %g\n",
           *btctau,*desiredT);
  }
```

```
  if (*bpctau > 0.0) {
    printf("Doing Berendsen pressure control with tau beta %g %g\n",
           *bpctau,*bpcbeta);
  }

  return;

}



void ReadAtoms(x,atomname,N)
int  *N;
struct vector *x;
char atomname[MAXAT][5];
{
  char comment[80],buf[120];
  int  i,type,n;
  FILE *fp;

  fp=fopen("atoms.in","r");

  if (fp==NULL) { puts("File atoms.in open failed"); exit(0); }

  fgets(buf,120,fp); sscanf(buf,"%d",N);

  if (*N > MAXAT) { puts("Error: MAXAT overflow, increase it."); exit(0); }

  fgets(comment,80,fp);
  printf("%s%d%s%40s\n","Reading in ",*N," atoms described as ",comment);

  for (i=1;i<= *N;i++) {

    fgets(buf,120,fp);
    n=sscanf(buf,"%s %lg %lg %lg %d",atomname[i],
           &(x[i].x),&(x[i].y),&(x[i].z),&type);

    if (n<5) {
        printf("Atom read in error for line %s n %d\n",buf,n);
        printf("%s %lg %lg %lg %d\n",atomname[i],x[i].x,x[i].y,x[i].z,type);
        puts("Atom read in error");
        exit(0);
    }

  }


}



void WriteAtoms(x,atomname,Ekin,Epot,N,time,size)

struct vector *x;
char atomname[MAXAT+1][5];
int  N;
double  *Ekin,*Epot,time;
struct vector size;

{

  char  comment[80];
```



```
  int  i,type;
  static FILE *fp;

  static int firsttime=1;

  if (firsttime) {
    fp=fopen("atoms.out","w");
    if (fp==NULL) { puts("File atoms.out open failed"); exit(0); }
    firsttime=0;
  }

  fprintf(fp," %d\n",N);

  fprintf(fp,"%s %11.3f %s %12.4f %12.4f %12.4f\n",
          "mdmorse atom output at time",
          time,"fs boxsize ",size.x,size.y,size.z);

  for(i=1;i<=N;i++) {
    type=1;
    fprintf(fp,"%2s %12.6f%12.6f%12.6f%11.6f%11.6f\n",
            atomname[i],x[i].x,x[i].y,x[i].z,Ekin[i],Epot[i]);
  }

  fflush(fp);

}
```

```
#include "global.h"

void SetTemperature(v,N,m,T)
struct vector  *v;
int  N;
double T,m;
{

  //
  // Subroutine sets the temperature of the atoms in the system
  // to T assuming a Maxwell-Boltzmann velocity distribution
  //


  // stuff removed

}


void GetTemperature(v,N,m,T)

struct vector  *v;
int  N;
double  *T,m;
{
  //
  // Subroutine gets the temperature of the atoms in the system
  //

  double  Ekinsum,gaussianrand();
  int  i;

  // Get sum of kinetic energies
  Ekinsum=0;
  for (i=1;i<=N;i++) {

     Ekinsum=Ekinsum+v[i].x*v[i].x+v[i].y*v[i].y+v[i].z*v[i].z;

  }
  // Do necessary unit transformations to get E in J
  Ekinsum=0.5*m*u*Ekinsum*vunit*vunit;
  // and get temperature using E=3/2 N k T
  *T = Ekinsum/(1.5*N*kB);

}

void GetEnergies(v,N,m,Ekinsum,Epotsum,Etot,Ekin,Epot)

struct vector  *v;
double  *Ekinsum,*Epotsum,*Etot,m,*Ekin,*Epot;
int  N;
{
  double  help1;
  int  i;

  *Ekinsum=0.0; *Epotsum=0.0;
  help1=0.5*m*u*vunit*vunit/e;
  for (i=1;i<=N;i++) {
     Ekin[i]=help1*(v[i].x*v[i].x+v[i].y*v[i].y+v[i].z*v[i].z);
     *Ekinsum = *Ekinsum+Ekin[i];
```



```
     *Epotsum = *Epotsum+Epot[i];
  }
  *Etot = *Ekinsum+*Epotsum;

}




//-----------------------------------------------------------------------------

double uniformrand(seed)
long *seed;

  // -----------------------------------------------------------
  // Park-Miller "minimal" Random number generator
  // uniform distribution ]0,1[ . See Numerical Recipes ch. 7.0
  // -----------------------------------------------------------
{
  long IA,IM,IQ,IR,MASK;
  double AM,ans;
  long  k;

  IA=16807; IM=2147483647; AM=1.0/IM; IQ=127773; IR=2836; MASK=123459876;

  *seed= (*seed)^MASK;
  k=(*seed)/IQ;
  *seed = IA*((*seed)-k*IQ)-IR*k;
  if (*seed < 0) *seed = *seed+IM;
  ans=AM*(*seed);
  *seed= (*seed)^MASK;

  return(ans);
}



double gaussianrand(seed)
int  *seed;

  // ----------------------------------------------------------
  // Random numbers with normal (Gaussian) distribution.
  // Mean is 0 and standard deviation is 1
  // ----------------------------------------------------------
{

  // stuff removed

}
```

```c
#include "global.h"

void UpdateNeighbourlist(x,N,size,periodic,neighbourlist,rskincut)

struct vector   *x;
int N,*neighbourlist;
struct vector   size;
struct ivector  periodic;
double   rskincut;


// Subroutine constructs a Verlet neighbour list in the
// following format:
//
// nngbr1           Number of neighbours for atom 1
// ingbr1,1         Index of neighbour 1 of atom 1
// ingbr2,1         Index of neighbour 2 of atom 1
// ....
// ingbrnngbr1,1   Index of neighbour nngbr1 of atom 1
// nngbr2           Number of neighbours for atom 2
// ...
//
// Note that the first index is 1, not 0 !!
{

  int nneighbourstot;


  // Stuff removed

  printf("%s %13.5f %s\n","Neighbour list update found",
       1.0*nneighbourstot/N,"neighbours per atom");

}
```

```
#include "global.h"

void Solve1(x,v,a,N,size,periodic,deltat)

struct vector   *x,*v,*a;
int         N;
double   deltat;
struct vector   size;
struct ivector  periodic;

{

  // Stuff removed

}


void Solve2(v,a,N,deltat,btctau,T,desiredT)

struct vector *v,*a;
int        N;
double  deltat;
double T,desiredT,btctau;
{

  // Stuff removed

}
```

```c
#include "global.h"

void GetForces(x,a,N,size,periodic,m,neighbourlist,rpotcut,
               morseDe,morsealpha,morseRe,Epot,virial)

struct vector   *x,*a;
int    N,*neighbourlist;
double   m,rpotcut;
struct vector   size;
struct ivector  periodic;
double morseDe,morsealpha,morseRe,*Epot;
double *virial;
{

  // Stuff removed

}
```