```
F90= f90 -O

MODULES= modules.o
OBJECTS= main.o inout.o physical.o neighbourlist.o solve.o forces.o
TARGET= mdmorse


mdmorse:                $(OBJECTS) $(MODULES)
                        $(F90) -o $(TARGET) $(OBJECTS) $(MODULES)


main.o:                 main.f90 $(MODULES)
                        $(F90) -c main.f90

inout.o:                inout.f90 $(MODULES)
                        $(F90) -c inout.f90

physical.o:             physical.f90 $(MODULES)
                        $(F90) -c physical.f90

neighbourlist.o:        neighbourlist.f90 $(MODULES)
                        $(F90) -c neighbourlist.f90

solve.o:                solve.f90 $(MODULES)
                        $(F90) -c solve.f90

forces.o:               forces.f90 $(MODULES)
                        $(F90) -c forces.f90



modules.o:      modules.f90
                $(F90) -c modules.f90
```

```
!
! 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
!
! Simulation cell centered at the origin.
!

program mdmorse
  USE GLOBAL
  implicit none

  ! 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
  !
  type(vector)   :: x(MAXAT),v(MAXAT),a(MAXAT)
  real(double)   :: Epot(MAXAT),Ekin(MAXAT)
  character(5)   :: atomname(MAXAT)
  integer        :: neighbourlist(MAXAT*MAXNEIGHBOURS)

  ! 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)
  !
  real(double)   :: time,tmax,deltat,T,Ekinsum,Epotsum,Etot,m
  !
  integer        :: N,nneighbourlistupdate,itime,nmovieoutput
  real(double)   :: rpotcut,rskincut
  type(vector)   :: size
  type(ivector)  :: periodic

  ! Morse potential parameters
  real(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.
  real(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
  real(double) :: bpctau,bpcbeta,P,desiredP,virial,mu

  integer :: i

  print *,''
  print *,'---------------- mdmorse03 V1.2 ----------------------'
  print *,''

  !
  ! Initialize the simulation
  !
  call 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;

  call ReadAtoms(x,atomname,N)

  ! Factor 2 because of energy equipartition theorem
  call SetTemperature(v,N,m,2.0*initialT)
  ! Check that we got it right
  call GetTemperature(v,N,m,T)
  print *,'Initial atom temperature is ',T

  ! Initialize accelerations to 0
  do i=1,N
     a(i)%x=0.0; a(i)%y=0.0; a(i)%z=0.0;
  enddo

  ! Start main loop over times
  time=0.0
  itime=0
  P=0.0
  do
     if (mod(itime,nneighbourlistupdate)==0) then
        call UpdateNeighbourlist(x,N,size,periodic,neighbourlist,rskincut)
     endif

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

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

     call GetTemperature(v,N,m,T)

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

     ! Update time
     time=time+deltat

     ! Actual solution done, now compute result quantities

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

     ! Output pressure in units of kbar
     print '(A,F10.3,1X,F10.3,4F12.5)','ec ',time,T,Ekinsum/N,  &
          Epotsum/N,Etot/N,P
```

```
      if (mod(itime,nmovieoutput)==0) then
         print '(A,F10.3)','Outputting atom movie at t = ',time
         call WriteAtoms(x,atomname,Ekin,Epot,N,time,size)
      endif

      ! Check whether we are done
      if (time >= tmax) exit

      itime=itime+1

   end do

   ! Always output final energy and atom coordinates
   ! using high precision
   print '(A,F12.5,1X,F12.5,3F14.7)','ec ',time,T,Ekinsum/N,Epotsum/N,Etot/N

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

   ! Close movie file
   close(31)

end program mdmorse
```

```
module global
  implicit none

  integer :: seed

  !
  integer, parameter :: double=SELECTED_REAL_KIND(P=14,R=100)

  ! Define a vector type
  type vector
     real(double) :: x,y,z
  end type vector

  ! Define an integer vector type
  type ivector
     integer :: x,y,z
  end type ivector

  ! 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
  !
  real(double), parameter :: pi=3.14159265358979
  real(double), parameter :: e=1.6021892d-19
  real(double), parameter :: u=1.660565d-27
  real(double), parameter :: kB=1.380662d-23
  real(double), parameter :: lunit=1e-10
  real(double), parameter :: tunit=1e-15
  real(double), parameter :: vunit=lunit/tunit
  real(double), parameter :: aunit=vunit/tunit

  ! Set array sizes

  integer, parameter :: MAXAT=10000
  integer, parameter :: MAXNEIGHBOURS=100

end module global
```

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

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

  USE GLOBAL
  implicit none

  real(double)  :: initialT,m,tmax
  type(vector)  :: size
  type(ivector) :: periodic
  real(double)  :: deltat,rpotcut,rskincut,morseDe,morsealpha,morseRe
  integer       :: nneighbourlistupdate,nmovieoutput
  real(double)  :: btctau,desiredT,bpctau,bpcbeta,desiredP

  integer :: i,ios
  character(80) :: line
  character(40) :: string
  real(double) :: x

  open(20,file='mdmorse.in',status='old',iostat=ios)

  if (ios /= 0) STOP 'Open error for file mdmorse.in'

  ! 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
  do
     i=i+1

     ! First read in line in whole.
     read(20,fmt='(A80)',iostat=ios) line
     ! Check for probable end-of-file
     if (ios < 0) exit

     ! If not parameter line, cycle
     if (line(1:1) /= '-') cycle

     ! Attempt to parse line into command and value
     ! using internal formatted read
     read(unit=line,fmt=*,iostat=ios) string,x
     if (ios > 0) then
        print *,'ERROR: Data read in error on line',i
        stop 'Invalid parameter file'
```



```
     endif

     ! From here on line should be in variable format

     ! Look for variable-defining strings
     if (string=='-initialT') then
        initialT=x
     else if (string=='-desiredT') then
        desiredT=x
     else if (string=='-btctau') then
        btctau=x
     else if (string=='-mass') then
        m=x
     else if (string=='-xsize') then
        size%x=x
     else if (string=='-ysize') then
        size%y=x
     else if (string=='-zsize') then
        size%z=x
     else if (string=='-periodicx') then
        periodic%x=int(x+0.5)
     else if (string=='-periodicy') then
        periodic%y=int(x+0.5)
     else if (string=='-periodicz') then
         periodic%z=int(x+0.5)
     else if (string=='-seed') then
        seed=int(x+0.5)
     else if (string=='-tmax') then
        tmax=x
     else if (string=='-deltat') then
        deltat= x
     else if (string=='-nupdate') then
        nneighbourlistupdate=int(x+0.5)
     else if (string=='-rpotcut') then
        rpotcut=x
     else if (string=='-rskincut') then
        rskincut=x
     else if (string=='-morseDe') then
        morseDe=x
     else if (string=='-morsealpha') then
        morsealpha=x
     else if (string=='-morseRe') then
         morseRe=x
     else if (string=='-bpctau') then
         bpctau=x
     else if (string=='-bpcbeta') then
         bpcbeta=x
     else if (string=='-desiredP') then
         desiredP=x
     else if (string=='-nmovieoutput') then
        nmovieoutput=int(x+0.5)
     else
        print '(A,A)','Unknown parameter',string
        stop 'Parameter read in error'
     endif

     print '(A,A16,A,G13.6)','Read in parameter ',string,' value',x

  enddo

  close(20)
```

```
  ! Print out some general information about read results:

  print '(A,3I2)','Using periodics (1=on, 0=off)',       &
       periodic%x,periodic%y,periodic%z

  print '(A,3F12.6)','Morse potential parameters: De alpha Re', &
       morseDe,morsealpha,morseRe

  print '(A,I8,A)','Movie output selected every',nmovieoutput,' steps'

  if (btctau > 0.0) then
     print '(/,A,2F10.3,/)','Doing Berendsen temperature control with tau T', &
          btctau,desiredT
  endif

  if (bpctau > 0.0) then
     print '(/,A,2F10.3,/)','Doing Berendsen pressure control with tau beta', &
          bpctau,bpcbeta
  endif

  return

end subroutine ReadParams


subroutine ReadAtoms(x,atomname,N)
  USE GLOBAL
  implicit none

  integer :: N
  type(vector) :: x(*)
  character(5) atomname(*)

  character(80) :: comment
  integer :: i,ios,type

  open(30,file='atoms.in',status='old',iostat=ios)

  if (ios/=0) STOP 'File atoms.in open failed'

  read(30,fmt=*) N

  if (N > MAXAT) STOP 'Error: MAXAT overflow, increase it.'

  read(30,fmt='(A80)') comment
  print '(A,I7,A,A40)','Reading in ',N,' atoms described as ',comment

  do i=1,N

     read(30,fmt=*,iostat=ios) atomname(i),x(i)%x,x(i)%y,x(i)%z,type

     if (ios/=0) then
        print *,'Atom read in error for atom',i
        print *,atomname(i),x(i)%x,x(i)%y,x(i)%z,type
        STOP ' Atom read in error'
     endif

  enddo


end subroutine ReadAtoms
```



```
subroutine WriteAtoms(x,atomname,Ekin,Epot,N,time,size)
  USE GLOBAL
  implicit none

  type(vector) :: x(*)
  character(5) :: atomname(*)
  integer :: N
  real(double) :: Ekin(*),Epot(*),time
  type(vector) :: size

  character(80) :: comment
  integer :: i,ios,type

  logical, save :: firsttime=.true.

  if (firsttime) then
     open(31,file='atoms.out',status='replace',iostat=ios)
     if (ios/=0) STOP 'File atoms.out open failed'

     firsttime=.false.
  endif


  write(31,fmt='(I10)') N

  write(31,fmt='(A,F11.3,A,3F12.4)') 'mdmorse atom output at time ', &
       time,' fs boxsize ',size%x,size%y,size%z

  do i=1,N
     type=1
     write(31,fmt='(A2,1X,3F12.6,2F11.6)') atomname(i),x(i)%x,x(i)%y,x(i)%z, &
          Ekin(i),Epot(i)
  enddo

end subroutine WriteAtoms
```

```
subroutine SetTemperature(v,N,m,T)
  USE GLOBAL
  implicit none

  !
  ! Subroutine sets the temperature of the atoms in the system
  ! to T assuming a Maxwell-Boltzmann velocity distribution
  !
  type(vector) :: v(*)
  integer :: N
  real(double) :: T,m

  ! stuff removed

end subroutine SetTemperature


subroutine GetTemperature(v,N,m,T)
  USE global
  implicit none

  !
  ! Subroutine gets the temperature of the atoms in the system
  !
  type(vector) :: v(*)
  integer :: N
  real(double) :: T,m

  real(double) :: Ekinsum,gaussianrand
  integer :: i

  ! Get sum of kinetic energies
  Ekinsum=0
  do i=1,N

     Ekinsum=Ekinsum+v(i)%x*v(i)%x+v(i)%y*v(i)%y+v(i)%z*v(i)%z

  enddo
  ! 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)

  return

end subroutine GetTemperature

subroutine GetEnergies(v,N,m,Ekinsum,Epotsum,Etot,Ekin,Epot)
  USE GLOBAL
  implicit none

  type(vector) :: v(*)
  real(double) :: Ekinsum,Epotsum,Etot,m,Ekin(*),Epot(*)
  integer :: N

  real(double) :: help1
  integer :: i

  Ekinsum=0.0; Epotsum=0.0;
  help1=0.5*m*u*vunit*vunit/e;
```



```
  do i=1,N
     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)
  enddo
  Etot=Ekinsum+Epotsum

  return

end subroutine GetEnergies


!-------------------------------------------------------------------------------

double precision function uniformrand(seed)
  implicit none

  ! ------------------------------------------------------------
  ! Park-Miller "minimal" Random number generator
  ! uniform distribution ]0,1[ . See Numerical Recipes ch. 7.0
  ! ------------------------------------------------------------

  integer :: seed,IA,IM,IQ,IR,MASK
  double precision :: AM
  integer :: k

  parameter (IA=16807,IM=2147483647,AM=1.0/IM,IQ=127773,IR=2836,MASK=123459876)

  seed=ieor(seed,MASK)
  k=seed/IQ
  seed=IA*(seed-k*IQ)-IR*k
  if (seed < 0) seed=seed+IM
  uniformrand=AM*seed
  seed=ieor(seed,MASK)

  return

end function uniformrand



double precision function gaussianrand(seed)
  implicit none

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

  integer :: seed

  ! Stuff removed

end function gaussianrand
```

```
subroutine UpdateNeighbourlist(x,N,size,periodic,neighbourlist,rskincut)
  USE GLOBAL
  implicit none

  ! 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
  ! ...


  type(vector)  :: x(*)
  integer       :: N,neighbourlist(*)
  type(vector)  :: size
  type(ivector) :: periodic
  real(double)  :: rskincut

  integer :: nneighbourstot



  ! Stuff removed


  print '(A,G13.5,A)','Neighbour list update found ',      &
       1.0*nneighbourstot/N,' neighbours per atom'

end subroutine UpdateNeighbourlist
```

```
subroutine Solve1(x,v,a,N,size,periodic,deltat)
  USE GLOBAL
  implicit none

  type(vector)  :: x(*),v(*),a(*)
  integer       :: N
  real(double)  :: deltat
  type(vector)  :: size
  type(ivector) :: periodic

  ! stuff removed

end subroutine Solve1


subroutine Solve2(v,a,N,deltat,btctau,T,desiredT)
  USE GLOBAL
  implicit none

  type(vector) :: v(*),a(*)
  integer      :: N
  real(double) :: deltat
  real(double) :: T,desiredT,btctau

  ! stuff removed

end subroutine Solve2
```

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

  USE GLOBAL
  implicit none

  type(vector)  :: x(*),a(*)
  integer       :: N,neighbourlist(*)
  real(double)  :: m,rpotcut
  type(vector)  :: size
  type(ivector) :: periodic
  real(double)  :: morseDe,morsealpha,morseRe,Epot(*)
  real(double)  :: virial

  ! Stuff removed

end subroutine GetForces
```