```
/* [[pr_03_2 - neighbor list and leapfrog]] (C) 2004  D. C. Rapaport */
/*************************************************************************
  This software is copyright material accompanying the book
  "The Art of Molecular Dynamics Simulation", 2nd edition,
  by D. C. Rapaport, published by Cambridge University Press (2004).
**************************************************************************/

#include "in_mddefs.h"

typedef struct {
  VecR r, rv, ra;
} Mol;

Mol *mol;
VecR region, vSum;
VecI initUcell;
real deltaT, density, rCut, temperature, timeNow, uSum, velMag, vvSum;
Prop kinEnergy, totEnergy;
int moreCycles, nMol, randSeed, stepAvg, stepCount, stepEquil, stepLimit;
VecI cells;
int *cellList;
real dispHi, rNebrShell;
int *nebrTab, nebrNow, nebrTabFac, nebrTabLen, nebrTabMax;
real virSum;
Prop pressure;
real kinEnInitSum;
int stepInitlzTemp;

NameList nameList[] = {
  NameR (deltaT),
  NameR (density),
  NameI (initUcell),
  NameI (nebrTabFac),
  NameI (randSeed),
  NameR (rNebrShell),
  NameI (stepAvg),
  NameI (stepEquil),
  NameI (stepInitlzTemp),
  NameI (stepLimit),
  NameR (temperature),
};

int main (int argc, char **argv)
{
  GetNameList (argc, argv);
  PrintNameList (stdout);
  SetParams ();
  SetupJob ();
  moreCycles = 1;
  while (moreCycles) {
    SingleStep ();
    if (stepCount >= stepLimit) moreCycles = 0;
  }
}

void SingleStep ()
{
  ++ stepCount;
  timeNow = stepCount * deltaT;
  LeapfrogStep (1);
  ApplyBoundaryCond ();
  if (nebrNow) {
    nebrNow = 0;
    dispHi = 0.;
    BuildNebrList ();
  }
  ComputeForces ();
  LeapfrogStep (2);
  EvalProps ();
  if (stepCount < stepEquil) AdjustInitTemp ();
  AccumProps (1);
  if (stepCount % stepAvg == 0) {
    AccumProps (2);
    PrintSummary (stdout);
    AccumProps (0);
  }
}
```

```
void SetupJob ()
{
  AllocArrays ();
  InitRand (randSeed);
  stepCount = 0;
  InitCoords ();
  InitVels ();
  InitAccels ();
  AccumProps (0);
  kinEnInitSum = 0.;
  nebrNow = 1;
}

void SetParams () {
  rCut = pow (2., 1./6.);
  VSCopy (region, 1. / pow (density, 1./3.), initUcell);
  nMol = VProd (initUcell);
  velMag = sqrt (NDIM * (1. - 1. / nMol) * temperature);
  VSCopy (cells, 1. / (rCut + rNebrShell), region);
  nebrTabMax = nebrTabFac * nMol;
}

void AllocArrays () {
  AllocMem (mol, nMol, Mol);
  AllocMem (cellList, VProd (cells) + nMol, int);
  AllocMem (nebrTab, 2 * nebrTabMax, int);
}


void BuildNebrList ()
{
  VecR dr, invWid, rs, shift;
  VecI cc, m1v, m2v, vOff[] = OFFSET_VALS;
  real rrNebr;
  int c, j1, j2, m1, m1x, m1y, m1z, m2, n, offset;

  rrNebr = Sqr (rCut + rNebrShell);
  VDiv (invWid, cells, region);
  for (n = nMol; n < nMol + VProd (cells); n ++) cellList[n] = -1;
  DO_MOL {
    VSAdd (rs, mol[n].r, 0.5, region);
    VMul (cc, rs, invWid);
    c = VLinear (cc, cells) + nMol;
    cellList[n] = cellList[c];
    cellList[c] = n;
  }
  nebrTabLen = 0;
  for (m1z = 0; m1z < cells.z; m1z ++) {
    for (m1y = 0; m1y < cells.y; m1y ++) {
      for (m1x = 0; m1x < cells.x; m1x ++) {
        VSet (m1v, m1x, m1y, m1z);
        m1 = VLinear (m1v, cells) + nMol;
        for (offset = 0; offset < N_OFFSET; offset ++) {
          VAdd (m2v, m1v, vOff[offset]);
          VZero (shift);
          VCellWrapAll ();
          m2 = VLinear (m2v, cells) + nMol;
          DO_CELL (j1, m1) {
            DO_CELL (j2, m2) {
              if (m1 != m2 || j2 < j1) {
                VSub (dr, mol[j1].r, mol[j2].r);
                VVSub (dr, shift);
                if (VLenSq (dr) < rrNebr) {
                  if (nebrTabLen >= nebrTabMax)
                     ErrExit (ERR_TOO_MANY_NEBRS);
                  nebrTab[2 * nebrTabLen] = j1;
                  nebrTab[2 * nebrTabLen + 1] = j2;
                  ++ nebrTabLen;
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
void ComputeForces () {
  VecR dr;
  real fcVal, rr, rrCut, rri, rri3, uVal;
  int j1, j2, n;

  rrCut = Sqr (rCut);
  DO_MOL VZero (mol[n].ra);
  uSum = 0.;
  virSum = 0.;
  for (n = 0; n < nebrTabLen; n ++) {
    j1 = nebrTab[2 * n];
    j2 = nebrTab[2 * n + 1];
    VSub (dr, mol[j1].r, mol[j2].r);
    VWrapAll (dr);
    rr = VLenSq (dr);
    if (rr < rrCut) {
      rri = 1. / rr;
      rri3 = Cube (rri);
      fcVal = 48. * rri3 * (rri3 - 0.5) * rri;
      uVal = 4. * rri3 * (rri3 - 1.) + 1.;
      VVSAdd (mol[j1].ra, fcVal, dr);
      VVSAdd (mol[j2].ra, - fcVal, dr);
      uSum += uVal;
      virSum += fcVal * rr;
    }
  }
}

void LeapfrogStep (int part) {
  int n;

  if (part == 1) {
    DO_MOL {
      VVSAdd (mol[n].rv, 0.5 * deltaT, mol[n].ra);
      VVSAdd (mol[n].r, deltaT, mol[n].rv);
    }
  } else {
    DO_MOL VVSAdd (mol[n].rv, 0.5 * deltaT, mol[n].ra);
  }
}

void ApplyBoundaryCond () {
  int n;
  DO_MOL VWrapAll (mol[n].r);
}

void AdjustInitTemp () {
  real vFac;
  int n;

  kinEnInitSum += kinEnergy.val;
  if (stepCount % stepInitlzTemp == 0) {
    kinEnInitSum /= stepInitlzTemp;
    vFac = velMag / sqrt (2. * kinEnInitSum);
    DO_MOL VScale (mol[n].rv, vFac);
    kinEnInitSum = 0.;
  }
}

void InitCoords () {
  VecR c, gap;
  int n, nx, ny, nz;

  VDiv (gap, region, initUcell);
  n = 0;
  for (nz = 0; nz < initUcell.z; nz ++) {
    for (ny = 0; ny < initUcell.y; ny ++) {
      for (nx = 0; nx < initUcell.x; nx ++) {
        VSet (c, nx + 0.5, ny + 0.5, nz + 0.5);
        VMul (c, c, gap);
        VVSAdd (c, -0.5, region);
        mol[n].r = c;
        ++ n;
      }
    }
  }
}
```

```
void InitVels ()
{
  int n;

  VZero (vSum);
  DO_MOL {
    VRand (&mol[n].rv);
    VScale (mol[n].rv, velMag);
    VVAdd (vSum, mol[n].rv);
  }
  DO_MOL VVSAdd (mol[n].rv, - 1. / nMol, vSum);
}


void InitAccels ()
{
  int n;

  DO_MOL VZero (mol[n].ra);
}


void EvalProps ()
{
  real vv, vvMax;
  int n;

  VZero (vSum);
  vvSum = 0.;
  vvMax = 0.;
  DO_MOL {
    VVAdd (vSum, mol[n].rv);
    vv = VLenSq (mol[n].rv);
    vvSum += vv;
    vvMax = Max (vvMax, vv);
  }
  dispHi += sqrt (vvMax) * deltaT;
  if (dispHi > 0.5 * rNebrShell) nebrNow = 1;
  kinEnergy.val = 0.5 * vvSum / nMol;
  totEnergy.val = kinEnergy.val + uSum / nMol;
  pressure.val = density * (vvSum + virSum) / (nMol * NDIM);
}


void AccumProps (int icode)
{
  if (icode == 0) {
    PropZero (totEnergy);
    PropZero (kinEnergy);
    PropZero (pressure);
  } else if (icode == 1) {
    PropAccum (totEnergy);
    PropAccum (kinEnergy);
    PropAccum (pressure);
  } else if (icode == 2) {
    PropAvg (totEnergy, stepAvg);
    PropAvg (kinEnergy, stepAvg);
    PropAvg (pressure, stepAvg);
  }
}


void PrintSummary (FILE *fp)
{
  fprintf (fp,
     "%5d %8.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f\n",
     stepCount, timeNow, VCSum (vSum) / nMol, PropEst (totEnergy),
     PropEst (kinEnergy), PropEst (pressure));
  fflush (fp);
}


#include "in_rand.c"
#include "in_errexit.c"
#include "in_namelist.c"
```