

## Appendix

Since the thrust of the homework problems is for the student to write, debug, and run 'homemade' programs, we will not provide a compendium of simulational software. Nonetheless, to provide some aid to the student in the learning process, we will offer a few programs that demonstrate some of the basic steps in a Monte Carlo simulation. We do wish to make the reader aware, however, that these program do not have all of the 'bells and whistles' which one might wish to introduce in a serious study but are merely simple programs that can be used to test the students' approach.

### Program 1 *Test a random number generator*

Note, as an exercise the student may wish to insert other random number generators or add tests to this simple program.

```
c*****
c This program is used to perform a few very simple tests of a random
c number generator. A congruential generator is being tested
c*****
      Real*8 Rnum(100000) , Rave, R2Ave, Correl, SDev
      Integer Iseed, num
      open(Unit=1, file='result_testrng_02')
      PMod = 2147483647.0D0
      DMax = 1.0D0/PMod
c*****
c Input
c*****
      write(*, 800)
800  format('enter the random number generator seed ')
      read(*, 921) Iseed
921  format(i5)
      write(*, 801) Iseed
      write(1, 801) Iseed
801  format(' The random number seed is ', I8)
      write(*, 802)
802  format('enter the number of random numbers to be generated')
      read(*, 921) num
      write(*, 803) num
      write(1, 803) num
803  format('number of random numbers to be generated = ', i8)
c*****
c Initialize variables, vectors
c*****
```

```

      do 1 i=2,10000
1     Rnum(i)=0.0D0
      Rave=0.D0
      Correl=0.0D0
      R2Ave=0.0D0
      SDev=0.0D0
c*****
c Calculate random numbers
c*****
      Rnum(1)=Iseed*DMax
      Write(*,931) Rnum(1),Iseed
      Do 10 i=2,num
          Rnum(i)=cong16807(Iseed)
          if (num.le.100) write(*,931) Rnum(i),Iseed
931  format(f10.5,i15)
10   continue
      Rave=Rnum(1)
      R2Ave=Rnum(1)**2
      Do 20 i=2,num
          Correl=Correl+Rnum(i)*Rnum(i-1)
          Rave=Rave+Rnum(i)
20   R2Ave=R2Ave+Rnum(i)**2
      Rave=Rave/num
      SDev=Sqrt((R2Ave/num-Rave**2)/(num-1))
      Correl=Correl/(num-1)-Rave*Rave
c*****
c Output
c*****
      write(*,932) Rave,SDev,Correl
932  format('Ave. random number = ',F10.6,' +/- ',F10.6,
1     /' '\nn''-correlation = ' F10.6)
      write(1,932) Rave,SDev,Correl
999  format(f12.8)
      close(1)
      stop
      end

      FUNCTION Cong16807(ISeed)
c*****
c This is a simple congruential random number generator
c*****
      INTEGER ISeed,IMod
      REAL*8 RMod,PMod,DMax
      RMod = DBLE(ISeed)
      PMod = 2147483647.0D0
      DMax = 1.0D0/PMod
      RMod = RMod*16807.0D0
      IMod = RMod*DMax
      RMod = RMod - PMod*IMod
      cong16807=rmod*DMax
      Iseed=Rmod
      RETURN
      END

```

**Program 2 A good routine for generating a table of random numbers**

```

C*****
C This program uses the R250/R521 combined generator described in:
C A. Heuer, B. Duenweg and A.M. Ferrenberg, Comp. Phys. Comm. 103, 1
C 1997). It generates a vector, RanVec, of length RanSize 31-bit random
C integers. Multiply by RMaxI to get normalized random numbers. You
C will need to test whether RanCnt will exceed RanSize. If so, call
C GenRan again to generate a new block of RanSize numbers. Always
C remember to increment RanCnt when you use a number from the table.
C*****
      IMPLICIT NONE
      INTEGER RanSize, Seed, I, RanCnt, RanMax
      PARAMETER(RanSize = 10000)
      PARAMETER( RMaxI = 2147483647 )
      INTEGER RanVec(RanSize), Z1(250+RanSize), Z2(521+RanSize)
      REAL*8 RMaxI
      PARAMETER ( RMaxI = 1.0D0/(1.0D0*RanMax) )
      COMMON/MyRan/RanVec, Z1, Z2, RanCnt
      SAVE
      Seed = 432987111
C*****
C Initialize the random number generator.
C*****
      CALL InitRan(Seed)*
C*****
c If the 10 numbers we need pushes us past the end of the RanVec vector,
C call GenRan. Since we just called InitRan, RanCnt = RanSize we must
c call it here.
C*****
      IF ((RanCnt + 10) .GT. RanSize) THEN
C** Generate RanSize numbers and reset the RanCnt counter to 1
          Call GenRan
      END IF
      Do I = 1, 10
          WRITE(*, *) RanVec(RanCnt + I - 1), RMaxI*RanVec(RanCnt + I - 1)
      End Do
      RanCnt = RanCnt + 10
C*****
C Check to see if the 10 numbers we need will push us past the end
C of the RanVec vector. If so, call GenRan.
C*****
      IF ((RanCnt + 10) .GT. RanSize) THEN
C** Generate RanSize numbers and reset the RanCnt counter to 1
          Call GenRan
      END IF
      Do I = 1, 10
          WRITE(*, *) RanVec(RanCnt + I - 1), RMaxI*RanVec(RanCnt + I - 1)
      End Do
      RanCnt = RanCnt + 10
      END

```

```

SUBROUTINE InitRan(Seed)
C*****
C Initialize the R250 and R521 generators using a congruential generator
C to set the individual bits in the 250/521 numbers in the table. The
C R250 and R521 are then warmed-up by generating 1000 numbers.
C*****
IMPLICIT NONE
REAL*8 RMaxI, RMod, PMod
INTEGER RanMax, RanSize
PARAMETER ( RanMax = 2147483647 )
PARAMETER (RanSize = 100000)
PARAMETER ( RMaxI = 1.0D0/(1.0D0*RanMax) )
INTEGER Seed, I, J, K, IMod, IBit
INTEGER RanVec (RanSize), Z1(250+RanSize), Z2(521+RanSize)
INTEGER RanCnt
COMMON/MyRan/RanVec, Z1, Z2, RanCnt
SAVE
RMod = DBLE(Seed)
PMod = DBLE(RanMax)
C*****
C Warm up a congruential generator
C*****
Do I = 1, 1000
  RMod = RMod*16807.0D0
  IMod = RMod/PMod
  RMod = RMod - PMod*IMod
End Do
C*****
C Now fill up the tables for the R250 & R521 generators: This
C requires random integers in the range 0-> 2*31 1. Iterate a
C strange number of times to improve randomness.
C*****
Do I = 1, 250
  Z1(I) = 0
  IBit = 1
  Do J = 0, 30
    Do K = 1, 37
      RMod = RMod*16807.0D0
      IMod = RMod/PMod
      RMod = RMod - PMod*IMod
    End Do
    C** Now use this random number to set bit J of X(I).
    IF (RMod .GT. 0.5D0*PMod) Z1(I) = IEOR(Z1(I), IBit)
    IBit = IBit*2
  End Do
End Do
Do I = 1, 521
  Z2(I) = 0
  IBit = 1
  Do J = 0, 30
    Do K = 1, 37
      RMod = RMod*16807.0D0
      IMod = RMod/PMod
      RMod = RMod - PMod*IMod
    End Do
    C** Now use this random number to set bit J of X(I).
    IF (RMod .GT. 0.5D0*PMod) Z2(I) = IEOR(Z2(I), IBit)
  End Do
End Do

```

```

IBit = IBit*2
End Do
End Do
C*****
C Perform a few iterations of the R250 and R521 random number generators
C to eliminate any effects due to 'poor' initialization.
C*****
  Do I = 1,1000
    Z1(I+250) = IEOR(Z1(I),Z1(I+147))
    Z2(I+521) = IEOR(Z2(I),Z2(I+353))
  End Do
  Do I = 1,250
    Z1(I) = Z1(I + 1000)
  End Do
  Do I = 1,521
    Z2(I) = Z2(I + 1000)
  End Do
C*****
C Set the random number counter to RanSize so that a proper checking
C code will force a call to GenRan in the main program.
C*****
  RanCnt = RanSize
  RETURN
  END

  SUBROUTINE GenRan
C*****
C Generate vector RanVec (length RanSize) of pseudo-random 31-bit
C integers.
C*****
  IMPLICIT NONE
  INTEGER RanSize,RanCnt,I
  PARAMETER(RanSize = 100000)
  INTEGER RanVec(RanSize),Z1(250+RanSize),Z2(521+RanSize)
  COMMON/MyRan/RanVec,Z1,Z2,RanCnt
  SAVE
C*****
C Generate RanSize pseudo-random numbers using the individual gen-
C erators
C*****
  Do I = 1,RanSize
    Z1(I+250) = IEOR(Z1(I),Z1(I+147))
    Z2(I+521) = IEOR(Z2(I),Z2(I+353))
  End Do
C*****
C Combine the R250 and R521 numbers and put the result into RanVec
C*****
  Do I = 1,RanSize
    RanVec(I) = IEOR(Z1(I+250),Z2(I+521))
  End Do
C*****
C Copy the last 250 numbers generated by R250 and the last 521 numbers
C from R521 into the working vectors (Z1), (Z2) for the next pass.
C*****
  Do I = 1,250
    Z1(I) = Z1(I + RanSize)
  End Do

```

```

      Do I = 1, 521
        Z2(I) = Z2(I + RanSize)
      End Do
C*****
C Reset the random number counter to 1.
C*****
      RanCnt = 1
      RETURN
      END

```

### Program 3 *The Hoshen–Kopelman cluster finding routine*

```

C*****
c lx, ly = lattice size along x, y
c ntrymax = number of lattices to be studied for each concentration
c iclmax = number of clusters (including those of 0 elements) found
c in a lattice configuration for a given concentration
c ioclmax = number of different cluster sizes found
c ns(1, j) = cluster size, j=1, ioclmax
c ns(2, j) = number of clusters of that size, j=1, ioclmax
c ninf = number of infinite clusters
c ninf/ntrymax = probability of infinite cluster
c
c For more details on the method, see:
c J. Hoshen and R. Kopelman, Phys. Rev. B14, 3428 (1976).
C*****
      Parameter (lxmax=500, lymax=500)
      Parameter (nmat=lxmax*lymax, nclustermax=nmat/2+1)
      Integer isiti (lxmax, lymax)
      Integer list (nmat), ncluster (nmat), nlabel (nclustermax)
      Integer ibott (lxmax), itop (lxmax), ileft (lymax), iright (lymax)
      Integer iperc (100), nsize (nclustermax), ns (2, nclustermax)
      Character*40 fout
C*****
c Input data
C*****
      read(5, *) lx
      read(5, *) ly
      read(5, *) fout
      if (lx.gt.lxmax) stop 'lx too big'
      if (ly.gt.lymax) stop 'ly too big'
C*****
c List of the sites
C*****
      num=0
      do j=1, lx
        do i=1, ly
          num=num+1
          isiti(i, j)=num
        enddo
      enddo
      nat=num
C*****
c Initialize
C*****
      ninf=0

```

```

iocl=0
ns(1,icl)=0
ns(2,icl)=0
do num=1,nat
  list(num)=0
  ncluster(num)=0
enddo
do icl=1,nclustermax
  nsize(icl)=0
enddo
open(unit=50,file=fout,status='unknown',form='formatted')
c*****
c Input spins
c*****
do iy=1,ly
  read(5,*) (list(isiti(ix,iy)),ix=1,lx)
enddo
c*****
c Analysis of the cluster
c*****
icl=0
if (list(1).eq.1) then
  icl=icl+1
  ncluster(1)=icl
  nlabel(icl)=icl
endif
do num=2,lx
  if (list(num).eq.1) then
    if (list(num-1).eq.1) then
      ivic1=ncluster(num-1)
      ilab1=nlabel(ivic1)
      ncluster(num)=ilab1
      icheck=1
    else
      icl=icl+1
      ncluster(num)=icl
      nlabel(icl)=icl
    endif
  endif
endif
enddo
do jj=1,ly-1
  num=jj*lx+1
  if (list(num).eq.1) then
    if (list(num-lx).eq.1) then
      ivic2=ncluster(num-lx)
      ilab2=nlabel(ivic2)
      ncluster(num)=ilab2
      icheck=1
    else
      icl=icl+1
      ncluster(num)=icl
      nlabel(icl)=icl
    endif
  endif
endif
do num=jj*lx+2,(jj+1)*lx
  if (list(num).eq.1) then
    if (list(num-1).eq.1) then

```

```

        ivic1=ncluster(num-1)
        ilab1=nlabel(ivic1)
        if (list(num-lx).eq.1) then
            ivic2=ncluster(num-lx)
            ilab2=nlabel(ivic2)
            imax=max(ilab1,ilab2)
            imin=min(ilab1,ilab2)
            ncluster(num)=imin
            nlabel(imax)=nlabel(imin)
            do kj=1,icl
                if (nlabel(kj).eq.imax) nlabel(kj)=imin
            enddo
            icheck=1
        else
            ncluster(num)=ilab1
            icheck=1
        endif
    endif
else
    if (list(num-lx).eq.1) then
        ivic2=ncluster(num-lx)
        ilab2=nlabel(ivic2)
        ncluster(num)=ilab2
        icheck=1
    else
        icl=icl+1
        ncluster(num)=icl
        nlabel(icl)=icl
    endif
endif
endif
endif
enddo
if (icheck.eq.0) then
    write(*,*) 'no possible percolation'
    go to 2000
endif
icheck=0
enddo
iclmax=icl
c*****
c Determination of the number of infinite clusters
c*****
io=0
do num=1,lx
    itest=0
    if (list(num).eq.1) then
        ilab=nlabel(ncluster(num))
        call conta(num,ilab,ibott,itest,io,lx)
    endif
enddo
iomax=io
in=0
do num=(ly-1)*lx+1,nat
    itest=0
    if (list(num).eq.1) then
        ilab=nlabel(ncluster(num))
        call conta(num,ilab,itop,itest,in,lx)
    endif
endif

```



```

enddo
inmax=in
il=0
do num=1,nat,lx
  itest=0
  if (list(num).eq.1) then
    ilab=nlabel(ncluster(num))
    call conta(num,ilab,ileft,itest,il,ly)
  endif
enddo
ilmax=il
ir=0
do num=lx,nat,lx
  itest=0
  if (list(num).eq.1) then
    ilab=nlabel(ncluster(num))
    call conta(num,ilab,iright,itest,ir,ly)
  endif
enddo
irmax=ir
nperc=0
nperc1=0
np=0
do ii=1,iomax
  do jj=1,inmax
    if (itop(jj).eq.ibott(ii)) then
      nperc=nperc+1
      np=np+1
      iperc(np)=nperc
    endif
  enddo
enddo
npmax=np
itest2=0
do ii=1,irmax
  do jj=1,ilmax
    if (ileft(jj).eq.iright(ii)) then
      do np=1,npmax
        if (ileft(jj).eq.iperc(np)) itest2=1
      enddo
      if (itest2.eq.0) nperc=nperc+1
    endif
  enddo
enddo
if (nperc.gt.0) nperc1=1
if (nperc.gt.0) ninf=ninf+1
call size(nat,iclmax,nsize,nlabel,ncluster,ns,iocl,
* nclustermax)
ioclmax=iocl
fl=1.0/float(nat)
do icl=1,ioclmax
  fl1=log(float(ns(1,icl)))
  fl2=log(float(ns(2,icl))*fl)
  write (50,*) ns(1,icl),ns(2,icl),float(ns(2,icl))*fl,fl1,fl2
enddo
write (*,*) 'Number of cluster sizes = ', ioclmax
write (*,*) 'Number of infinite clusters = ', ninf

```

```

2000 continue
      stop
      end

SUBROUTINE size(nat, iclmax, nsize, nlabel, ncluster, ns, iocl, nclmax)
  integer nlabel(nclmax), ncluster(nat), nsize(iclmax)
  integer ns(2, nclmax)
  do num=1, nat
    do ncl=1, iclmax
      if (nlabel(ncluster(num)).eq.ncl) nsize(ncl)=nsize(ncl)+1
    enddo
  enddo
  write(*,*) 'Number of clusters = ', iclmax
  do ncl=1, iclmax
    write(*,*) 'Cluster # ', ncl, ', size = ', nsize(ncl)
  enddo
  write(*,*) ''
  do ncl=1, iclmax
    if (nsize(ncl).gt.0) then
      if (iocl.eq.0) then
        iocl=iocl+1
        ns(1, iocl)=nsize(ncl)
        ns(2, iocl)=1
      else
        itest3=0
        do i=1, iocl
          if (nsize(ncl).eq.ns(1, i)) then
            ns(2, i)=ns(2, i)+1
            itest3=1
          endif
        enddo
        if (itest3.eq.0) then
          iocl=iocl+1
          ns(1, iocl)=nsize(ncl)
          ns(2, iocl)=1
        endif
      endif
    endif
  enddo
  return
end

SUBROUTINE conta(num, ilab, icona, itest, io, ll)
  Integer icona(ll)
  if (io.eq.0) then
    io=io+1
    icona(io)=ilab
    itest=1
  else
    do ii=1, io
      if (ilab.eq.icona(ii)) itest=itest+1
    enddo
    if (itest.gt.1) stop 'error in icona'
    if (itest.eq.0) then
      io=io+1
      icona(io)=ilab
    endif
  endif

```

```

endif
return
end

SUBROUTINE ass(rint, rn, ipos, ll)
zero=1.d-6
do nn=1, ll
  rmax=nn*rint
  rmin=(nn-1)*rint
  if ((rn-rmin).ge.zero).and.((rn-rmax).lt.zero) then
    ipos=nn
    go to 100
  endif
enddo
100 return
end

```

#### Program 4 *The one-dimensional Ising model*

```

c*****
c This simple program performs a Monte Carlo simulation of a 1-dim
c Ising model with a periodic boundary. Parameters are inputted
c from the screen. Sweeps in either temperature or field can be run.
c Data output is to the screen and to a data file
c*****
  Logical new
  Real*4 Jint
  Common/index/ nrun
  Common/sizes/n, nsq
  Common/param/beta, betah
  Common/inparm/ temp, field, Jint
  open(Unit=1, file='result_1d_Ising_MCB.dat')
  new=.true.
  write(*, 900)
  write(1, 900)
900  format(' Monte Carlo simulation of the d=1 Ising model')
  Iseed=12345
  write(*, 2929) Iseed
2929 format(' random number seed is ', I8)
  Inrg=ran(iseed)
c*****
c enter input parameters:
c*****
  write(*, 905)
905  format(' enter n [length of the chain]')
  read(*, 910) n
910  format(i10)
  write(*, 912)
912  format(' enter the coupling constant')
  read(*, 920) Jint
  write(*, 915)
915  format(' enter the initial temperature')
  read(*, 920) temp
920  format(f20.6)
  write(*, 925)
925  format(' enter the temperature increment')
  read(*, 920) tempi

```

```

write(*,930)
930 format(' enter the initial magnetic field')
read(*,920) field
write(*,935)
935 format(' enter the magnetic field increment')
read(*,920) fieldi
write(*,940)
940 format(' enter the number of runs')
read(*,910) numrun
write(*,945)
945 format(' enter number of MC-steps ')
read(*,910) mcstps
write(*,950)
950 format(' enter the number of steps discarded for equilibrium')
read(*,910) ntoss
nint=1
write(*,955) n,mcstps,ntoss
write(1,955) n,mcstps,ntoss
955 format('/' 1-dimensional Ising chain of length',1x,i3/1x,i9,' mc-
*steps/site with ',1x,i8,' mcs/s discarded to reach equilibrium '/')
write(*,960) Jint
write(1,960) Jint
960 format(' coupling constant = ',f8.4)
ncount=mcstps/nint
temp=temp-tempi
field=field-fieldi
do 1111 jrun=1,numrun
  nrun=jrun
  call results(-1)
  temp=temp+tempi
  field=field+fieldi
c*****
c Check the temperature to prevent underflow/overflow
c*****
  if(abs(temp).lt.1.0e-5) then
    write(*,6666)
6666  format(' Stop the simulation; this temperature is too cold!')
    goto 6677
  endif
  beta=Jint/temp
  betah=field/temp
c*****
c Calculate flipping probabilities
c*****
  call carlo(new)
  if(ntoss.ge.1) call monte(ntoss,Irng)
c*****
c Plot lattice after equilibration
c*****
  write(*,970)
970  format('New run: Picture of the lattice after equilibration:')
  call picture
c*****
c Do a simulation and calculate results
c*****
  do 1 jmc=1,ncount
    call monte(nint,Irng)

```

```

        call core(n)
        call results(0)
1    continue
c*****
c Now, output results and a snapshot of the lattice
c*****
        call results(1)
        write(*,975)
975  format('A picture of the lattice at the end of the run:')
        call picture
        write(*,980)
980  format(//)
1111 continue
6677 call results(2)
        close(1)
        stop
        end

        SUBROUTINE core(n)
c*****
c Calculate the energy and magnetization for a configuration
c*****
        Integer*2 Ispin(80)
        Real*8 e(20),wn
        Common/corrs/ e
        Common/spins/Ispin
        nel=0
        nh1=0
        jm=n
        do 1 j=1,n
            nel=nel+Ispin(j)*Ispin(jm)
            nh1=nh1+Ispin(j)
            jm=j
1    continue
        wn=1.0d0/(n)
        e(1)=nel*wn
        e(2)=nh1*wn
        return
        end

        SUBROUTINE picture
c*****
c Produce a snapshot of the lattice
c*****
        Integer*2 Ispin(80)
        Character plus,minus,ising(80)
        Common/spins/Ispin
        Common/sizes/n,nsq
        data plus,minus/'+', '-' /
        do 2 j=1,n
            ising(j)=plus
            if(Ispin(j).ne.1) ising(j)=minus
2    continue
        write(*,200) (ising(k),k=1,n)
200  format(1x,80a1)
        return
        end

```

```

SUBROUTINE monte (mcstps, Irng)
c*****
c Perform a Monte Carlo step/site
c*****
Integer*2 Ispin(80)
Integer*2 neigh(20)
Real*4 prob(9,3), rn
Common/spins/Ispin
Common/sizes/n, nsq
Common/trans/ prob
nml=n-1
if (nml.eq.0) nml=1
do 1 mc=1, mcstps
  jmc=0
  do 2 jj=1, n
    j=n*RAN(Irng)+1.0e-06
    jp=j+1
    if (jp.gt.n) j=1
    jm=j-1
    if (jm.lt.1) jm=n
    rn=RAN(Irng)
    jmc=jmc+1
    nc=Ispin(j)
    n4=Ispin(jm)+Ispin(jp)
    n4=nc*n4+3
    nh=nc+2
    if (rn.gt.prob(n4, nh)) goto 6
    Ispin(j)=-nc
  6   continue
2   continue
1   continue
return
end

SUBROUTINE carlo (new)
c*****
c Calculate the table of flipping probabilities
c*****
Logical new
Integer*2 Ispin(80)
Real*4 prob(9,3)
Common/spins/Ispin
Common/sizes/n, nsq
Common/trans/ prob
Common/param/beta, betah
nsq=n*n
if ((abs(betah).gt.30.0).or.(abs(beta).gt.30.0)) then
  write(*, 6666)
6666  format(' Stop the simulation; the temperature is too cold!')
  stop
endif
do 11 j=1, 5
  do 11 jh=1, 3
    prob(j, jh)=exp(-2.0*beta*(j-3)-2.0*betah*(jh-2))
11  continue
if (.not.new) return
new=.false.

```

```

do 2 j=1,n
  Ispin(j)=1
2  continue
  write(*,950)
950 format('initial state:')
  call picture
  write(*,960)
960 format(//)
  return
  end

SUBROUTINE results(111)
c*****
c Output results
c*****
  Real*8 e(99), ee(99), am(99), amm(99), am4(99), U(99)
  Real*8 dam(99), de(99), spheat(99), cor(20), wnum
  Real temper(99), fields(99)
  Common/inparm/ temp,field,Jint
  Common/sizes/n,nsq
  Common/index/1
  Common/corrs/cor
  if(111) 1,2,3
1  continue
  e(1)=0.0d0
  ee(1)=0.0d0
  am(1)=0.0d0
  amm(1)=0.0d0
  am4(1)=0.0d0
  num=0
  return
2  continue
  num=num+1
  e(1)=e(1)+cor(1)
  ee(1)=ee(1)+cor(1)*cor(1)
  am(1)=am(1)+cor(2)
  amm(1)=amm(1)+cor(2)*cor(2)
  am4(1)= am4(1)+cor(2)**4
  return
3  continue
  if(111.gt.1) goto 4
  write(*,99)
99  format(/t4,'T',t10,'H',t17,'U4',t25,'E',t31,'E*E',
  * t39,'dE**2',t50,'M',t58,'M*M',t66,'dM**2',t76,'C')
  wnum=1.0d0/num
  temper(1)=temp
  fields(1)=field
  e(1)=e(1)*wnum
  ee(1)=ee(1)*wnum
  am(1)=am(1)*wnum
  amm(1)=amm(1)*wnum
  am4(1)=am4(1)*wnum
  de(1)=ee(1)-e(1)*e(1)
  dam(1)=amm(1)-am(1)*am(1)
  U(1)=1.0d0-am4(1)/(3.0d0*amm(1)**2)
  fn=1.0d0*n
  spheat(1)=fn*de(1)/(temper(1)**2)

```

```

write(*,100) temper(1),fields(1),U(1),e(1),ee(1),de(1),
* am(1),amm(1),dam(1),spheat(1)
return
4 continue
write(*,900)
900 format('Summary of the results:')
write(*,99)
write(1,99)
do 55 j=1,1
write(*,100) temper(j),fields(j),U(j),e(j),ee(j),de(j),
* am(j),amm(j),dam(j),spheat(j)
write(1,100) temper(j),fields(j),U(j),e(j),ee(j),de(j),
* am(j),amm(j),dam(j),spheat(j)
100 format(2f6.3, 3f8.4, f8.4, f9.5, f9.5, f9.5, f7.3)
55 continue
return
end

```

### Program 5 *The Bond Fluctuation Method*

Note, this program contains yet another random number generator.

```

c*****
c This program simulates a simple 3-dim lattice model for polymers
c using the athermal bond-fluctuation method. For more details see:
c I. Carmesin and K. Kremer, Macromolecules 21, 2878 (1988).
c*****
Implicit none
Integer seed, nrmeas, mcswait
Character*50 infile, outfile, outres
include ``model.common``
include ``lattice.common``
write(*,*) 'input file for the old configuration:'
read(*, '(a50)') infile
write(*,*) infile
write(*,*) 'output file for the new configuration:'
read(*, '(a50)') outfile
write(*,*) outfile
write(*,*) 'output file for measurements:'
read(*, '(a50)') outres
write(*,*) outres
write(*,*) 'time lapse between two measurements:'
read(*,*) mcswait
write(*,*) mcswait
write(*,*) 'number of measurements:'
read(*,*) nrmeas
write(*,*) nrmeas
write(*,*) 'seed for the random number generator:'
read(*,*) seed
write(*,*) seed
c*****
c Initialize the bond vectors
c*****
call bdibf
c*****
c Initialize the bond angles and index for the bond angles
c*****

```



```

      call aninbf
c*****
c Initialize the table for the allowed moves
c*****
      call inimove
c*****
c read in the configuration and initialize the lattice
c*****
      call bfin(infile)
c*****
c MC simulation part
c*****
      call bflsim(mcswait,nrmeas,seed,outres)
c*****
c write out the end configuration
c*****
      call bflout(outfile)
      end

      SUBROUTINE aninbf
c*****
c This program calculates the possible bond-angles
c*****
      Implicit none
      Real skalp(108,108), winkel(100), pi
      Integer indx(100), index, i, j, k, double, new(88), sawtest
      Logical test
      include 'model.common'
c*****
c Initializing the set of bond angles
c*****
      pi = 4.0 * atan(1.0)
      index = 1
      do 410 i=1,108
        do 410 j=1,108
          winkel(index) = 5.0
          test = .false.
          sawtest = (bonds(i,1)+bonds(j,1))**2 +
*                 (bonds(i,2)+bonds(j,2))**2 +
*                 (bonds(i,3)+bonds(j,3))**2
          if(sawtest.ge.4) then
            test = .true.
            skalp(i,j) = bonds(i,1)*bonds(j,1) +
*                   bonds(i,2)*bonds(j,2) +
*                   bonds(i,3)*bonds(j,3)
            skalp(i,j) = skalp(i,j) / (bl(i)*bl(j))
            skalp(i,j) = min(skalp(i,j),1.0)
            skalp(i,j) = max(skalp(i,j),-1.0)
            skalp(i,j) = pi - acos(skalp(i,j))
            do 411 k=1,index
              if(abs(skalp(i,j)-winkel(k)).le.0.001) then
                test = .false.
                angind(i,j) = k
              endif
            411 continue
          if(test) then
            winkel(index) = skalp(i,j)
          endif
        410 continue
      410 continue

```

```

        angind(i,j) = index
        index = index + 1
        winkel(index) = 5.0
    endif
else
    angind(i,j) = 100
endif
410 continue
do 417 i=1,108
    do 417 j=1,108
        if(angind(i,j).eq.100) angind(i,j) = index
417 continue
call indexx(index,winkel,indx)
do 412 i=1,index
    angles(i) = winkel(indx(i))
    new(indx(i)) = i
412 continue
do 413 i=1,108
    do 413 j=1,108
        angind(i,j) = new(angind(i,j))
413 continue
return
end

SUBROUTINE bdibf
c*****
c This subroutine creates the allowed bond-set and passes it back.
c*****
    Implicit none
    Integer max, ipegel, i, j, k, index, ind
    Integer startvec(6,3), zielvec(50,3), testb(3), sumvec(3)
    Integer dumvec(50,3), bondnr, newbond(3), dummy
    Logical test, foundbond
    include 'model.common'
c*****
c INITIALIZING POSSIBLE BONDVECTORS
c*****
    startvec(1,1) = 2
    startvec(1,2) = 0
    startvec(1,3) = 0
    startvec(2,1) = 2
    startvec(2,2) = 1
    startvec(2,3) = 0
    startvec(3,1) = 2
    startvec(3,2) = 1
    startvec(3,3) = 1
    startvec(4,1) = 2
    startvec(4,2) = 2
    startvec(4,3) = 1
    startvec(5,1) = 3
    startvec(5,2) = 0
    startvec(5,3) = 0
    startvec(6,1) = 3
    startvec(6,2) = 1
    startvec(6,3) = 0
    max = 0
    do 210 i=1,6

```

```

ind = 1
do 211 j=1,2
  do 212 k=1,3
    zielvec(ind,1) = startvec(i,1)
    zielvec(ind,2) = startvec(i,2)
    zielvec(ind,3) = startvec(i,3)
    ind = ind + 1
    zielvec(ind,1) = startvec(i,1)
    zielvec(ind,2) = startvec(i,2)
    zielvec(ind,3) = - startvec(i,3)
    ind = ind + 1
    zielvec(ind,1) = startvec(i,1)
    zielvec(ind,2) = - startvec(i,2)
    zielvec(ind,3) = startvec(i,3)
    ind = ind + 1
    zielvec(ind,1) = startvec(i,1)
    zielvec(ind,2) = - startvec(i,2)
    zielvec(ind,3) = - startvec(i,3)
    ind = ind + 1
    zielvec(ind,1) = - startvec(i,1)
    zielvec(ind,2) = startvec(i,2)
    zielvec(ind,3) = startvec(i,3)
    ind = ind + 1
    zielvec(ind,1) = - startvec(i,1)
    zielvec(ind,2) = startvec(i,2)
    zielvec(ind,3) = - startvec(i,3)
    ind = ind + 1
    zielvec(ind,1) = - startvec(i,1)
    zielvec(ind,2) = - startvec(i,2)
    zielvec(ind,3) = startvec(i,3)
    ind = ind + 1
    zielvec(ind,1) = - startvec(i,1)
    zielvec(ind,2) = - startvec(i,2)
    zielvec(ind,3) = - startvec(i,3)
    ind = ind + 1
    dummy = startvec(i,1)
    startvec(i,1) = startvec(i,2)
    startvec(i,2) = startvec(i,3)
    startvec(i,3) = dummy
212 continue
dummy = startvec(i,1)
startvec(i,1) = startvec(i,2)
startvec(i,2) = dummy
211 continue
dumvec(1,1) = zielvec(1,1)
dumvec(1,2) = zielvec(1,2)
dumvec(1,3) = zielvec(1,3)
ipegel = 2
do 213 k=1,48
  index = 1
  test = .false.
333 if((.not.test).and.(index.lt.ipegel)) then
  test = ((zielvec(k,1).eq.dumvec(index,1)).and.
*        (zielvec(k,2).eq.dumvec(index,2))).and.
*        (zielvec(k,3).eq.dumvec(index,3)))
  index = index + 1
goto 333

```

```

endif
if(.not.test) then
  dumvec(ipegel,1) = zielvec(k,1)
  dumvec(ipegel,2) = zielvec(k,2)
  dumvec(ipegel,3) = zielvec(k,3)
  ipegel = ipegel + 1
endif
213 continue
do 214 j=1,ipegel-1
  bonds(max+j,1) = dumvec(j,1)
  bonds(max+j,2) = dumvec(j,2)
  bonds(max+j,3) = dumvec(j,3)
214 continue
max = max + ipegel - 1
210 continue
do 220 i=1,108
  bl2(i) = bonds(i,1)**2 + bonds(i,2)**2 + bonds(i,3)**2
  bl(i) = sqrt(bl2(i))
220 continue
return
end

SUBROUTINE bflin(infile)
c*****
c This subroutine reads in an old configuration. The first line of the
c configuration file contains the number of chains and degree of poly-
c merization. The chain conformations are stored in consecutive lines:
c One line contains x, y and z coordinates of the start monomer of the
c chain, and the next lines each contain 10 integers which are the
c numbers of the bonds connecting adjacent monomers. For each chain
c the last bond number is 109, indicating a chain end without a bond.
c This works only for chains with length N=k*10. The coordinates of
c monomers 2 to N are then reconstructed from this information.
c*****
  Implicit none
  Character*50 infile
  Integer i, j, jj, k, kd, kk, xp, yp, zp, xpl, ypl, zpl, nb, base
  Include 'model.common'
  Include 'lattice.common'
  open(11,file=infile,form='formatted',status='old')
  read(11,*) nrchains, polym
  ntot = nrchains * polym
  nb = polym / 10
  do 1 j=1,nrchains
    base = polym * (j-1)
    read(11,*) monpos(base+1,1), monpos(base+1,2), monpos
      (base+1,3)
    do 2 jj = 0, nb-1
      read(11,*) (monbd(k+10*jj+base), k=1,10)
2  continue
  do 3 k=2, polym
  do 3 kd=1, 3
    monpos(base+k, kd) = monpos(base+k-1, kd) +
      * bonds(monbd(base+k-1), kd)
    monlatp(base+k, kd) = mod(monpos(base+k, kd), 1s) + 1
    if(monlatp(base+k, kd).le.0) then
      monlatp(base+k, kd) = monlatp(base+k, kd) + 1s

```

```

endif
3  continue
1  continue
   monbd(0) = 109
   monbd(ntot+1) = 109
c*****
c These are the arrays for the periodic boundary conditions.
c*****
   do 10 i=1,ls
      ip(i) = i+1
      ip2(i) = i+2
      im(i) = i-1
10  continue
   ip(ls) = 1
   ip2(ls-1) = 1
   ip2(ls) = 2
   im(1) = ls
c*****
c Now we initialize the lattice, setting all occupied vertices to unity
c*****
   do 4 j=1,ls
      do 4 k=1,ls
         do 4 kk=1,ls
            latt(j,k,kk) = 0
4  continue
   do 5 j=1,ntot
      xp = monlatp(j,1)
      yp = monlatp(j,2)
      zp = monlatp(j,3)
      xp1 = ip(xp)
      yp1 = ip(yp)
      zp1 = ip(zp)
      latt(xp,yp,zp) = 1
      latt(xp1,yp,zp) = 1
      latt(xp,yp1,zp) = 1
      latt(xp,yp,zp1) = 1
      latt(xp1,yp1,zp) = 1
      latt(xp1,yp,zp1) = 1
      latt(xp,yp1,zp1) = 1
      latt(xp1,yp1,zp1) = 1
5  continue
end

SUBROUTINE bflout(outfile)
c*****
c Stores the final configuration of the simulation into a configura-
c tion file for use as a start configuration for a continuation run.
c*****
Implicit none
Character*50 outfile
Integer j, jj, k, nb, base
include 'model.common'
open(13,file=outfile,form='formatted',status='unknown')
write(13,*) nrchains, polym
nb = polym / 10
do 1 j=1,nrchains
   base = polym*(j-1) + 1

```

```

      write(13,*) monpos(base,1),monpos(base,2),monpos(base,3)
      do 2 jj = 0,nb-1
        base = polym * (j-1) + 10 * jj
        write(13,'(10I4)') (monbd(k+base),k=1,10)
2      continue
1      continue
      end

```

```

      SUBROUTINE bflsim(mcswait,nrmeas,seed,outres)
c*****
c Performs the actual Monte Carlo simulation using jumps to nearest-
c neighbor sites as the only type of moves.
c*****
      Implicit none
      Double precision r2m,r4m,rg2m,rg4m,lm,l2m
      Double precision rgnorm,blnorm,accept
      Real u(97),c,cd,cm
      Integer mcswait,nrmeas,seed,dir
      Integer i97,j97,imeas,iwait,ind,mono,yp,yp,zp
      Integer xml,xp1,xp2,ym1,yp1,yp2,zml,zp1,zp2
      Iinteger newbl,newbr,testlat
      Logical test
      Character*50 outres
      include ``model.common``
      include ``lattice.common``
      Common/raset1/ u,c,cd,cm,i97,j97
      Common/static/ r2m,r4m,rg2m,rg4m,lm,l2m
      open(12,file=outres,form='formatted',status='unknown')
c*****
c Initialize the cumulative measurement variables.
c*****
      r2m = 0.0d0
      r4m = 0.0d0
      rg2m = 0.0d0
      rg4m = 0.0d0
      lm = 0.0d0
      l2m = 0.0d0
      accept = 0.0d0
c*****
c Initialize the random number generator
c*****
      call rmarin(seed)
c*****
c loop over the number of measurements we wish to perform.
c*****
      do 10 imeas=1,nrmeas
c*****
c loop over the number of Monte Carlo steps between two measurements
c*****
      do 20 iwait=1,mcswait
        call ranmar(rand,3*ntot)
        ind = 1
        mono = ntot * rand(ind) + 1
        dir = 6 * rand(ind+1) + 1
        newbl = move(monbd(mono-1),dir)
        newbr = move(monbd(mono),dir)

```

```

test = (newbl.eq.0).or.(newbr.eq.0)
if(.not.test) then
  xp = monlatp(mono,1)
  yp = monlatp(mono,2)
  zp = monlatp(mono,3)
  if(dir.eq.1) then
c*****
c jump in +x direction
c*****
  xp2 = ip2(xp)
  xp1 = ip(xp)
  yp1 = ip(yp)
  zp1 = ip(zp)
  testlat = latt(xp2,yp,zp) + latt(xp2,yp1,zp) +
  *      latt(xp2,yp,zp1) + latt(xp2,yp1,zp1)
  if(testlat.eq.0) then
c*****
c new monomer positions and new bonds
c*****
  monpos(mono,1) = monpos(mono,1) + 1
  monlatp(mono,1) = xp1
  monbd(mono-1) = newbl
  monbd(mono) = newbr
c*****
c set the newly occupied vertices to one and the old to zero.
c*****
  latt(xp2,yp,zp) = 1
  latt(xp2,yp1,zp) = 1
  latt(xp2,yp,zp1) = 1
  latt(xp2,yp1,zp1) = 1
  latt(xp,yp,zp) = 0
  latt(xp,yp1,zp) = 0
  latt(xp,yp,zp1) = 0
  latt(xp,yp1,zp1) = 0
  accept = accept + 1.0d0
  endif
endif
if(dir.eq.6) then
c*****
c jump in -x direction
c*****
  xm1 = im(xp)
  xp1 = ip(xp)
  yp1 = ip(yp)
  zp1 = ip(zp)
  testlat = latt(xm1,yp,zp) + latt(xm1,yp1,zp) +
  *      latt(xm1,yp,zp1) + latt(xm1,yp1,zp1)
  if(testlat.eq.0) then
c*****
c new monomer positions and new bonds
c*****
  monpos(mono,1) = monpos(mono,1) - 1
  monlatp(mono,1) = xm1
  monbd(mono-1) = newbl
  monbd(mono) = newbr

```

```

c*****
c set the newly occupied vertices to one and the old to zero.
c*****
      latt(xml,yp,zp) = 1
      latt(xml,yp1,zp) = 1
      latt(xml,yp,zp1) = 1
      latt(xml,yp1,zp1) = 1
      latt(xp1,yp,zp) = 0
      latt(xp1,yp1,zp) = 0
      latt(xp1,yp,zp1) = 0
      latt(xp1,yp1,zp1) = 0
      accept = accept + 1.0d0
    endif
  endif
  if(dir.eq.2) then
c*****
c jump in +y direction
c*****
      xp1 = ip(xp)
      yp1 = ip(yp)
      yp2 = ip2(yp)
      zp1 = ip(zp)
      testlat = latt(xp,yp2,zp) + latt(xp1,yp2,zp) +
      *          latt(xp,yp2,zp1) + latt(xp1,yp2,zp1)
      if(testlat.eq.0) then
c*****
c new monomer positions and new bonds
c*****
      monpos(mono,2) = monpos(mono,2) + 1
      monlatp(mono,2) = yp1
      monbd(mono-1) = newbl
      monbd(mono) = newbr
c*****
c set the newly occupied vertices to one and the old to zero.
c*****
      latt(xp,yp2,zp) = 1
      latt(xp1,yp2,zp) = 1
      latt(xp,yp2,zp1) = 1
      latt(xp1,yp2,zp1) = 1
      latt(xp,yp,zp) = 0
      latt(xp1,yp,zp) = 0
      latt(xp,yp,zp1) = 0
      latt(xp1,yp,zp1) = 0
      accept = accept + 1.0d0
    endif
  endif
  if(dir.eq.5) then
c*****
c jump in -y direction
c*****
      xp1 = ip(xp)
      yp1 = ip(yp)
      ym1 = im(yp)
      zp1 = ip(zp)
      testlat = latt(xp,ym1,zp) + latt(xp1,ym1,zp) +
      *          latt(xp,ym1,zp1) + latt(xp1,ym1,zp1)
      if(testlat.eq.0) then

```



```

c*****
c new monomer positions and new bonds
c*****
      monpos(mono,2) = monpos(mono,2) - 1
      monlatp(mono,2) = yml
      monbd(mono-1) = newbl
      monbd(mono) = newbr
c*****
c set the newly occupied vertices to one and the old to zero.
c*****
      latt(xp,yml,zp) = 1
      latt(xp1,yml,zp) = 1
      latt(xp,yml,zp1) = 1
      latt(xp1,yml,zp1) = 1
      latt(xp,yp1,zp) = 0
      latt(xp1,yp1,zp) = 0
      latt(xp,yp1,zp1) = 0
      latt(xp1,yp1,zp1) = 0
      accept = accept + 1.0d0
    endif
  endif
  if(dir.eq.3) then
c*****
c jump in +z direction
c*****
      xp1 = ip(xp)
      yp1 = ip(yp)
      zp1 = ip(zp)
      zp2 = ip2(zp)
      testlat = latt(xp,yp,zp2) + latt(xp1,yp,zp2) +
      *      latt(xp,yp1,zp2) + latt(xp1,yp1,zp2)
      if(testlat.eq.0) then
c*****
c new monomer positions and new bonds
c*****
      monpos(mono,3) = monpos(mono,3) + 1
      monlatp(mono,3) = zp1
      monbd(mono-1) = newbl
      monbd(mono) = newbr
c*****
c set the newly occupied vertices to one and the old to zero.
c*****
      latt(xp,yp,zp2) = 1
      latt(xp1,yp,zp2) = 1
      latt(xp,yp1,zp2) = 1
      latt(xp1,yp1,zp2) = 1
      latt(xp,yp,zp) = 0
      latt(xp1,yp,zp) = 0
      latt(xp,yp1,zp) = 0
      latt(xp1,yp1,zp) = 0
      accept = accept + 1.0d0
    endif
  endif
  if(dir.eq.4) then
c*****
c jump in -z direction
c*****

```

```

xp1 = ip(xp)
yp1 = ip(yp)
zp1 = ip(zp)
zml = im(zp)
testlat = latt(xp,yp,zml) + latt(xp1,yp,zml) +
*       latt(xp,yp1,zml) + latt(xp1,yp1,zml)
if(testlat.eq.0) then
c*****
c new monomer positions and new bonds
c*****
      monpos(mono,3) = monpos(mono,3) - 1
      monlatp(mono,3) = zml
      monbd(mono-1) = newbl
      monbd(mono) = newbr
c*****
c set the newly occupied vertices to one and the old to zero.
c*****
      latt(xp,yp,zml) = 1
      latt(xp1,yp,zml) = 1
      latt(xp,yp1,zml) = 1
      latt(xp1,yp1,zml) = 1
      latt(xp,yp,zp1) = 0
      latt(xp1,yp,zp1) = 0
      latt(xp,yp1,zp1) = 0
      latt(xp1,yp1,zp1) = 0
      accept = accept + 1.0d0
    endif
  endif
endif
ind = ind + 3
20 continue
c*****
c calculation of equilibrium properties
c*****
      call chainst
10 continue
c*****
c normalization of measurements
c*****
      rgnorm = nrchains*nrmeas
      blnorm = rgnorm*(polym-1)
      r2m = r2m / rgnorm
      r4m = r4m / rgnorm
      rg2m = rg2m / rgnorm
      rg4m = rg4m / rgnorm
      lm = lm / blnorm
      l2m = l2m / blnorm
      accept = accept / (1.0d0*ntot*mcswait*nrmeas)
c*****
c output of measured quantities
c*****
      write(12,*) 'Mean squared end-to-end distance: ',r2m
      write(12,*) 'Mean quartic end-to-end distance: ',r4m
      write(12,*) 'Mean squared radius of gyration: ',rg2m
      write(12,*) 'Mean quartic radius of gyration: ',rg4m
      write(12,*) 'Mean bond length: ',lm
      write(12,*) 'Mean squared bond length: ',l2m

```

```

write(12,*) 'Mean acceptance rate : ', accept
end

SUBROUTINE chainst
c*****
c This subroutine calculates some simple chain properties, e.g. the
c average end-to-end distance, radius of gyration and bond length.
c*****
Implicit none
Double precision r2m, r4m, rg2m, rg4m, lm, l2m
Double precision r2, r4, rg2, rg4, rcm(3), dpolym
Integer base, mon1, mon2, i, j
Common/static/r2m, r4m, rg2m, rg4m, lm, l2m
include 'model.common'
include 'lattice.common'
dpolym = polym*1.0d0
c*****
c Calculate 2nd and 4th moment of the end-to-end vector of the chains
c*****
do 10 i=1, nrchains
  mon1 = polym*(i-1) + 1
  mon2 = polym*i
  r2 = (monpos(mon2, 1) - monpos(mon1, 1)) ** 2 +
*      (monpos(mon2, 2) - monpos(mon1, 2)) ** 2 +
*      (monpos(mon2, 3) - monpos(mon1, 3)) ** 2
  r4 = r2 * r2
  r2m = r2m + r2
  r4m = r4m + r4
10 continue
c*****
c Calculate 2nd and 4th moments of the radius of gyration of the chains
c*****
do 20 i=1, nrchains
  rcm(1) = 0.0d0
  rcm(2) = 0.0d0
  rcm(3) = 0.0d0
  base = polym*(i-1)
  do 21 j=1, polym
    mon1 = base + j
    rcm(1) = rcm(1) + monpos(mon1, 1)
    rcm(2) = rcm(2) + monpos(mon1, 2)
    rcm(3) = rcm(3) + monpos(mon1, 3)
21 continue
  rcm(1) = rcm(1) / dpolym
  rcm(2) = rcm(2) / dpolym
  rcm(3) = rcm(3) / dpolym
  rg2 = 0.0d0

  do 22 j=1, polym
    mon1 = base + j
    rg2 = rg2 + (monpos(mon1, 1) - rcm(1)) ** 2 +
*              (monpos(mon1, 2) - rcm(2)) ** 2 +
*              (monpos(mon1, 3) - rcm(3)) ** 2
22 continue
  rg2 = rg2 / dpolym
  rg4 = rg2 * rg2
  rg2m = rg2m + rg2

```

```

          rg4m = rg4m + rg4
20      continue
c*****
c Calculate the 1st and 2nd moments of the bond length
c*****
      do 30 i=1,nrchains
          base = polym*(i-1)
          do 30 j=1,polym-1
              mon1 = base + j
              lm = lm + bl(monbd(mon1))
              l2m = l2m + bl2(monbd(mon1))
30      continue
      end

      SUBROUTINE INDEXX (N,ARRIN,INDX)
      DIMENSION ARRIN(N),INDX(N)
      DO 11 J=1,N
          INDX(J)=J
11      CONTINUE
          L=N/2+1
          IR=N
10      CONTINUE
          IF (L.GT.1) THEN
              L=L-1
              INDXT=INDX(L)
              Q=ARRIN(INDXT)
          ELSE
              INDXT=INDX(IR)
              Q=ARRIN(INDXT)
              INDX(IR)=INDX(1)
              IR=IR-1
              IF (IR.EQ.1) THEN
                  INDX(1)=INDXT
              RETURN
          ENDIF
          ENDIF
          I=L
          J=L+L
20      IF (J.LE.IR) THEN
          IF (J.LT.IR) THEN
              IF (ARRIN(INDX(J)).LT.ARRIN(INDX(J+1))) J=J+1
          ENDIF
          IF (Q.LT.ARRIN(INDX(J))) THEN
              INDX(I)=INDX(J)
              I=J
              J=J+J
          ELSE
              J=IR+1
          ENDIF
          GO TO 20
          ENDIF
          INDX(I)=INDXT
      GO TO 10
      END

```

```

SUBROUTINE inimove
C*****
Implicit none
Integer i, j, k, new(6,3)
Logical test
include ``model.common``
do 1 i=1,108
  new(1,1) = bonds(i,1) + 1
  new(1,2) = bonds(i,2)
  new(1,3) = bonds(i,3)
  new(2,1) = bonds(i,1)
  new(2,2) = bonds(i,2) + 1
  new(2,3) = bonds(i,3)
  new(3,1) = bonds(i,1)
  new(3,2) = bonds(i,2)
  new(3,3) = bonds(i,3) + 1
  new(4,1) = bonds(i,1)
  new(4,2) = bonds(i,2)
  new(4,3) = bonds(i,3) - 1
  new(5,1) = bonds(i,1)
  new(5,2) = bonds(i,2) - 1
  new(5,3) = bonds(i,3)
  new(6,1) = bonds(i,1) - 1
  new(6,2) = bonds(i,2)
  new(6,3) = bonds(i,3)
do 2 j=1,6
  test = .false.
  do 3 k=1,108
    test = (new(j,1).eq.bonds(k,1)).and.
*      (new(j,2).eq.bonds(k,2)).and.(new(j,3).eq.-
bonds(k,3))
    if(test) then
      move(i,j) = k
    else
      move(i,j) = 0
    endif
  3   continue
  2   continue
  1   continue
do 4 i=1,6
  move(109,i) = 109
  4   continue
end

SUBROUTINE RANMAR(RVEC, LEN)
C*****
C Random number generator proposed in: G. Marsaglia and A. Zaman,
C Ann. Appl. Prob. 1, 462 (1991). It generates a vector 'RVEC' of
C length 'LEN' OF pseudorandom numbers; the commonblock includes
C everything needed to specify the state of the generator.
C*****
DIMENSION RVEC(*)
COMMON/RASET1/U(97),C,CD,CM,I97,J97
DO 100 IVEC=1,LEN
  UNI = U(I97) - U(J97)
  IF(UNI.LT.0.) UNI = UNI + 1.
  U(I97) = UNI

```

```

I97 = I97 - 1
IF(I97.EQ.0) I97 = 97
J97 = J97 - 1
IF(J97.EQ.0) J97 = 97
C = C - CD
IF(C.LT.0.) C = C + CM
UNI = UNI - C
IF(UNI.LT.0.) UNI = UNI + 1.
RVEC(IVEC) = UNI
100 CONTINUE
RETURN
END

SUBROUTINE RMARIN(IJKL)
C*****
C Initializes RANMAR. The input value should be in the range:
C 0 <= IJKL <= 900 000 000 . To obtain the standard values in the
C MARSAGLIA - ZAMAN PAPER (I=12, J=34, K=56, L=78) PUT IJKL = 54217137
C*****
COMMON/RASET1/U(97),C,CD,CM,I97,J97
IJ = IJKL / 30082
KL = IJKL - IJ * 30082
I = MOD(IJ/177,177) + 2
J = MOD(IJ,177) + 2
K = MOD(KL/169,178) + 1
L = MOD(KL,169)
C WRITE(*,*) 'RANMAR INITIALIZED: ', IJKL, I, J, K, L
DO 2 II=1,97
S = 0.
T = 0.5
Do 3 JJ=1,24
M = MOD(MOD(I*J,179)*K,179)
I = J
J = K
L = MOD(53*L+1,169)
IF(MOD(L*M,64).GE.32) S = S + T
3 T = 0.5 * T
2 U(II) = S
C = 362436. / 16777216.
CD = 7654321. / 16777216.
CM = 16777213. / 16777216.
I97 = 97
J97 = 33
RETURN
END

c lattice.common
C*****
c ls = the linear size of the lattice in lattice constants
c nmax = the maximum number of monomers on the lattice
c maxch = the maximum number of chains.
C nmax, maxch > the requirements for the standard melt simulation: a
C volume fraction of 0.5 translates into 4000 monomers on the lattice
c Monomer positions and bonds are stored in arrays indexed by the
c number (n*k + j) for the j-th monomer in the k-th chain. Fake bonds
c lead to monomer 1 and from the last monomer so we won't have to

```

```

c distinguish between them and the other monomers (same for chain
ends) .
C*****
      Integer ls, nmax, maxch
      Parameter (ls=40, nmax=10001, maxch=500)
C*****
c For use with real random numbers and ranmar
C*****
      Real rand(3*nmax)
      Integer latt(ls,ls,ls), monbd(-1:nmax), monpos(nmax,3),
      * monlatp(nmax,3), ip(ls), ip2(ls), im(ls),
      * nrchains, polym, nrends, ntot
      Common/lattice/ rand, latt, monbd, monpos, monlatp, ip, ip2, im,
      * nrchains, polym, nrends, ntot

c      model.common
C*****
      Real angles(0:100),
      Real b1(108), b2(108)
      Integer bonds(110,3), angind(110,110), move(109,6)
      Common/model/ angles, b1, b2, bonds, angind, move

```