Algorithm for lMloopMC:

Inputs:

- Cubic lattice of dimensions (lx x ly x lz) = (7 x 7 x 20) consisting of nc= 36 polymer chains, each chain of bc=20 polymer beads, total sites are ns = 980.
- Bead attraction energy is E_b = -0.457, bead overlap energy is E_x = 200. Maximum moves maxm = 10^8.

Process (main()):

- Initialize variables and read bead positions for all chains from file chains.csv
- Initialize lattice occupation status using sstate()
- For each site index (sindex = 0 to sindex<ns),
    - If (lcs.sz[sindex] == 1): occupied state
        - Gives the details of occupied state (sindex, lcs.sx[sindex], lcs.sy[sindex], lcs.sz[sindex])
- For each chain (i=0 to i<nc),
    - For each bead (j=0 to j<bc):
        - Writes initial configuration to the file mchains.csv.
- Loop while (totm < maxm)
    - Randomly choose a chain (cnum) and attempt move using appropriate subroutines(fmeval(), lmeval(), kmoves()).
    - Calculate energy change (deltaE()) and apply metropolis algorithm (metrop()).
    - If accepted, Update bead coordinates.
- Increment totm by 1.
- Write intermediate bead coordinates to mchainsfn.csv for every 5 x 10^6 moves.
- Call sstate() to Check if a site has more than one bead(lcs.sz[k]>1) and write final bead configuration to mchainsc.csv
- return 0.

Outputs:

- Intermediate files for every 5 x 10^6 moves
- Final bead configuration to mchainsc.csv

Subroutines:

sstate():

- initialize chain index(i), bead index(j), and site index (k)
- For each site index (k = 0 to ns-1),
    - Initializes the lattice site status.
- For each chain (i = 0 to nc-1)
    - For each bead (j = 0 to bc-1)
        - Updates the occupation status and lattice site details

fmoves(cnum, r): cnum is chain number and r is direction index

- Computes the current orientation of the first(j=0) segment
- If move =1, check for the constraints and update new bead positions
- For move =0 or bead outside the box, reset to original position.

nmoves(cnum, r):

- Computes the current orientation of the last (j = 19) segment
- If move =1, check for the constraints and update new bead positions
- For move =0 or bead outside the box, reset to original position.

Kmoves(cnum, k):

- Determine the current orientation of k^{th} and (k-1) th segment
- If the dot product = 0, update a position and no move allowed if dot product $\neq 0$.
- Check for the constrains and reset the positions if outside the box

deltaE(olds, news):

- Initialize chain index (i), old neighbours sum (oldn), new neighbours sum (newn) and energy difference (Ediff).
- If the lattice site is already occupied (lcs.sz[news]>=1), returns high overlap energy (E_x = 200).
- Else,

        - compute the total number of neighbour occupations for old and new sites.
- returns energy difference = (newn – oldn) * E_b.

<u>fmeval</u>(cnum,findex,fcalc):

- makes a valid move for the first (j=0) bead via fmoves(cnum,r)
- calculates the new site index (scalc)
- evaluates the energy change for moving the chosen bead via deltaE(olds,news)

<u>lmeval</u>(cnum,lindex,lcalc):

- makes a valid move for the last (j=19) bead via nmoves(cnum,r)
- calculates the new site index (scalc)
- evaluates the energy associated for moving the chosen bead via deltaE(olds,news)

<u>metrop</u>(delE):

- Declare variables acc, r_{ij}, p_{ij}
- If delE <=0,
    - accept move without any probability (acc =1).
- Else,
    - generate r_{ij} = ran2(&seed) and
    - if p_{ij} (exp(-delE)) > r_{ij},
        - acc = 1 or else reject(acc=0).
- return acc

<u>accmov</u>(cnum,bnum,pcalc,scalc,vec mpos):

- Once the move has been accepted, decrements old lattice, lcs.sz[pcalc]-=1.
- Updates the beads coordinates.
- Increment new lattice site, lcs.sz[scalc]+=1.