```
#include <iostream>
#include <iomanip>
#include <math.h>
#include <stdlib.h>
#include <fstream>
#include <ctime>
#include <complex>
#include <cstdio>
#include <string>
#include "../MASTER-FILES/AtomParams.h"
using namespace std;

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////

void xAxisRotation(AtomParams **Atom, double rotationAngle, int nAtoms)
{
  struct vect tempPosition, position;

  for (int i=0; i<nAtoms; i++)
  {
    position = Atom[i]->getPosition();

    tempPosition.x = position.x;
    tempPosition.y = position.y*cos(rotationAngle) - position.z*sin(rotationAngle);
    tempPosition.z = position.y*sin(rotationAngle) + position.z*cos(rotationAngle);

    Atom[i]->setPosition( tempPosition );
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////

void yAxisRotation(AtomParams **Atom, double rotationAngle, int nAtoms)
{
  struct vect tempPosition, position;

  for (int i=0; i<nAtoms; i++)
  {
    position = Atom[i]->getPosition();

    tempPosition.x = position.x*cos(rotationAngle) - position.z*sin(rotationAngle);
    tempPosition.y = position.y;
    tempPosition.z = position.x*sin(rotationAngle) + position.z*cos(rotationAngle);

    Atom[i]->setPosition( tempPosition );
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////

void zAxisRotation(AtomParams **Atom, double rotationAngle, int nAtoms)
{
  struct vect tempPosition, position;

  for (int i=0; i<nAtoms; i++)
  {
    position = Atom[i]->getPosition();

    tempPosition.x = position.x*cos(rotationAngle) - position.y*sin(rotationAngle);
    tempPosition.y = position.x*sin(rotationAngle) + position.y*cos(rotationAngle);
    tempPosition.z = position.z;

    Atom[i]->setPosition( tempPosition );
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

void Translate(AtomParams **Atom, struct vect translationVector, int nAtoms)
{
  struct vect tempPosition, position;

  for (int i=0; i<nAtoms; i++)
  {
    position = Atom[i]->getPosition();
    tempPosition = position + translationVector;
    Atom[i]->setPosition( tempPosition );
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

int main(int argc, char *argv[])
{
    ////////////////
    // USER INPUT //
    ////////////////
  if (argc != 3) {
    cout << "Must have 2 command line arguments (nCarbon, nQuarter)!" << endl;
  }

  int nCarbon = atoi(argv[1]);
```

```
int nQuarter = atoi(argv[2]);

float n1, n2;
double latticeSpacing, domainAngle;
if (nCarbon == 6) {

  n1 = 11.0 + float(nQuarter)*0.25;
  n2 = 3.0 + float(nQuarter)*0.25;

  if (nQuarter == 0) {
    latticeSpacing = 31.401;
    domainAngle = 11.742;
  } else if (nQuarter == 1) {
    latticeSpacing = 32.414;
    domainAngle = 12.331;
  } else if (nQuarter == 2) {
    latticeSpacing = 33.430;
    domainAngle = 12.885;
  } else if (nQuarter == 3) {
    latticeSpacing = 34.450;
    domainAngle = 13.407;
  } else if (nQuarter == 4) {
    latticeSpacing = 35.472;
    domainAngle = 13.898;
  } else {}

} else if (nCarbon == 10) {

  n1 = 15.0 + float(nQuarter)*0.25;
  n2 = 1.0 + float(nQuarter)*0.25;

  if (nQuarter == 0) {
    latticeSpacing = 38.182;
    domainAngle = 3.198;
  } else if (nQuarter == 1) {
    latticeSpacing = 39.135;
    domainAngle = 3.901;
  } else if (nQuarter == 2) {
    latticeSpacing = 40.095;
    domainAngle = 4.571;
  } else if (nQuarter == 3) {
    latticeSpacing = 41.059;
    domainAngle = 5.209;
  } else if (nQuarter == 4) {
    latticeSpacing = 42.028;
    domainAngle = 5.818;
  } else {}

} else {}

int nAtomsPerTSB = 54+6*(3*nCarbon+1);
int nAtomsTotal = nAtomsPerTSB;
```

```
  // Create Instance of the class AtomParams
AtomParams *Atom[nAtomsTotal];

  ////////////////////////////////////////////
  // Read TSB input files (from Gaussian09) //
  ////////////////////////////////////////////
stringstream inStream1;
inStream1 << "../cpp-input/tsb35-c" << nCarbon << "_cpp-input.dat";
ifstream in1(inStream1.str().c_str());

cout << endl;
cout << "TSB input file to be read: " << inStream1.str() << endl;

string tempAtomName, tempAtomType, tempElementType;
struct vect tempPosition;
double tempCharge;

for (int i=0; i<nAtomsPerTSB; i++)
{
  in1 >> tempAtomName >> tempAtomType >> tempPosition.x >> tempPosition.y >> tempPosition.z >>
   tempElementType >> tempCharge;
  Atom[i] = new AtomParams(tempAtomName, tempAtomType, tempPosition, tempElementType,
  tempCharge);
}

cout << "TSB input file read and stored in class AtomParams/Atom." << endl;
cout << endl;

  // Create TSB output PDB files
stringstream outStream1;
outStream1 << "tsb35-c" << nCarbon << "_0" << nQuarter << "-quarter_TSBA-PDB.pdb";
ofstream OUT1(outStream1.str().c_str());
OUT1 << setiosflags(ios::fixed) << setprecision(3);

stringstream outStream2;
outStream2 << "tsb35-c" << nCarbon << "_0" << nQuarter << "-quarter_TSBB-PDB.pdb";
ofstream OUT2(outStream2.str().c_str());
OUT2 << setiosflags(ios::fixed) << setprecision(3);

cout << "TSB output files created: " << outStream1.str() << endl;
cout << "and " << outStream2.str() << endl;
cout << endl;

  ////////////////////////////
  // Build lattice vectors //
  ////////////////////////////
struct vect a1, a2, a3, b1, b2, b3;
double a = sqrt(3.0)*1.42;
double triAngle = 60.0*(PI/180.0);

  // Define graphene lattice vectors
a1.x = a;
a1.y = 0.0;
```

```
a1.z = 0.0;
a2.x = a*cos(triAngle);
a2.y = a*sin(triAngle);
a2.z = 0.0;
a3 = (1.0/3.0)*(a1+a2);

  // Define TSB lattice vectors
b1 = n1*a1 + n2*a2;
b2 = (-1.0)*n2*a1 + (n1+n2)*a2;
b3 = (-1.0/3.0)*(b1+b2);

double Bmag = 4.0*sqrt(b1.x*b1.x + b1.y*b1.y + b1.z*b1.z);

  // Calculate center of mass for the TSB molecule
double mass;
double totalMass=0.0;

struct vect position;
struct vect CoM;
CoM.x=0.0;
CoM.y=0.0;
CoM.z=0.0;

for (int i=0; i<nAtomsPerTSB; i++)
{
  mass = Atom[i]->getMass();

  position = Atom[i]->getPosition();
  Atom[i]->setPosition(position);

  CoM.x += mass * position.x;
  CoM.y += mass * position.y;
  CoM.z += mass * position.z;

  totalMass += mass;
}

CoM = (1.0/totalMass)*CoM;

Translate(Atom, (-1.0)*CoM, nAtomsPerTSB);
double rotationAngle = 60.0*(PI/180.0);
zAxisRotation(Atom, rotationAngle, nAtomsPerTSB);


////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////

struct vect hexagonCenter;
hexagonCenter.x = 0.5*a;
hexagonCenter.y = 0.5*a/sqrt(3.0);
hexagonCenter.z = 3.366;

  //go to a reasonable distance above graphene sheet, centered on the first graphene carbon
```

```
Translate(Atom, hexagonCenter, nAtomsPerTSB);

  //define number of TSB molecule "rows" in directions of b1 and b2
int nMolecules1 = 4;
int nMolecules2 = 4;

struct vect comTSBA, comTSBB;

int moleculeCount = 0;
  // Write TSBA molecules
OUT1 << "CRYST1   " << Bmag << "   " << Bmag << "  100.000   " << domainAngle << "   " <<
domainAngle << "   " << domainAngle << "           P 1            1" << endl;
for (int i=0; i<nMolecules1; i++)
{
  for (int j=0; j<nMolecules2; j++)
  {
    for (int k=0; k<nAtomsPerTSB; k++)
    {
     position = Atom[k]->getPosition();
     position = position + i*b1 + j*b2;
     Atom[k]->setPosition(position);

     OUT1 << "ATOM " << setw(6) << k + moleculeCount*nAtomsPerTSB + 1;
     OUT1 << " " << setw(4) << Atom[k]->getAtomName() << " TSB    " << setw(3) << moleculeCount
     +1 << "     ";
     OUT1 << setw(7) << position.x << " " << setw(7) << position.y << " " << setw(7) <<
     position.z << "   ";
     OUT1 << "0.00  0.00      " << setw(4) << "TSBA" << endl;

     if (i==0 && j==0)
     {
   mass = Atom[k]->getMass();
   position = Atom[k]->getPosition();
   Atom[k]->setPosition(position);

   comTSBA.x += mass * position.x;
   comTSBA.y += mass * position.y;
   comTSBA.z += mass * position.z;

   totalMass += mass;
     }

     position = Atom[k]->getPosition();
     position = position - i*b1 - j*b2;
     Atom[k]->setPosition(position);
    }
    moleculeCount++;
  }
}

comTSBA = (1.0/totalMass)*comTSBA;

  // Place molecule on origin by subtracting: (r_i - r_com)
```

```
  Translate(Atom, (-1.0)*CoM, nAtomsPerTSB);
    // Rotate by 60 degrees, interdigtating the rest of the TSB molecules
  zAxisRotation(Atom, rotationAngle, nAtomsPerTSB);
    // Replace molecule at (r_i + r_com)
  Translate(Atom, CoM, nAtomsPerTSB);

  moleculeCount = 0;
    // Write rotated molecules, using the TSB basis vector b3
  OUT2 << "CRYST1   " << Bmag << "   " << Bmag << "  100.000   " << domainAngle << "   " <<
  domainAngle << "   " << domainAngle << "          P 1           1" << endl;
  for (int i=0; i<nMolecules1; i++)
  {
    for (int j=0; j<nMolecules2; j++)
    {
      for (int k=0; k<nAtomsPerTSB; k++)
      {
       position = Atom[k]->getPosition();
       position = position + i*b1 + j*b2 + b3;
       Atom[k]->setPosition(position);

       OUT2 << "ATOM " << setw(6) << k + moleculeCount*nAtomsPerTSB + 1;
       OUT2 << " " << setw(4) << Atom[k]->getAtomName() << " TSB    " << setw(3) << moleculeCount
       +1 << "     ";
       OUT2 << setw(7) << position.x << " " << setw(7) << position.y << " " << setw(7) <<
       position.z << "   ";
       OUT2 << "0.00  0.00      " << setw(4) << "TSBB" << endl;

       if (i==0 && j==0)
       {
     mass = Atom[k]->getMass();
     position = Atom[k]->getPosition();
     Atom[k]->setPosition(position);

     comTSBB.x += mass * position.x;
     comTSBB.y += mass * position.y;
     comTSBB.z += mass * position.z;

     totalMass += mass;
       }

       position = Atom[k]->getPosition();
       position = position - i*b1 - j*b2 - b3;
       Atom[k]->setPosition(position);
      }
      moleculeCount++;
    }
  }

  comTSBB = (1.0/totalMass)*comTSBB;

  return 0;
}
```