

```

#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;

int countGrapheneAtoms (struct vect a1, struct vect a2, struct vect a3, struct vect b1, struct
vect b2, struct vect b3)
{
    struct vect B1, B2, B12, C1, C2, C12;

    // Define TSB super-lattice vectors
    B1 = 4.0*b1;
    B2 = 4.0*b2;
    B12 = B1 + B2;

    // Define boundaries of TSB super-lattice vectors (purely for calculation)
    C1 = B1 + 2.0*b3;
    C2 = B2 + 2.0*b3;
    C12 = B12 + 2.0*b3;

    double slopeC1 = (C1.y - 2.0*b3.y) / (C1.x - 2.0*b3.x);
    double slopeC2 = (C2.y - 2.0*b3.y) / (C2.x - 2.0*b3.x);
    double yInterceptC1 = C1.y - slopeC1*C1.x;
    double yInterceptC2 = C2.y - slopeC2*C2.x;

    double yIntercept1C12 = C12.y - slopeC1*C12.x;
    double yIntercept2C12 = C12.y - slopeC2*C12.x;

    // Initialize graphene basis vectors, where "1" and "2" denote basis atoms of the graphene
    unit cell
    struct vect positionG1 = 2.0*b3;
    struct vect positionG2 = 2.0*b3;

    // Initialize a sufficiently large grid to scan for atoms inside the boundaries
    int maxUnitCells1 = 500;
    int maxUnitCells2 = 500;
    int nAtomsPerLayer = 0;

    // Calculate atomic positions, counting nAtomsPerLayer
    for (int i=-maxUnitCells1; i<maxUnitCells1; i++)
    {
        for (int j=-maxUnitCells2; j<maxUnitCells2; j++)
        {
            positionG1 = i*a1 + j*a2 + a3;
            positionG2 = i*a1 + j*a2 + 2.0*a3;
        }
    }
}

```

```

double xLowerLimitG1 = (1.0/slopeC2)*( positionG1.y - yInterceptC2 );
double yLowerLimitG1 = slopeC1*positionG1.x + yInterceptC1;

double xUpperLimitG1 = (1.0/slopeC2)*( positionG1.y - yIntercept2C12);
double yUpperLimitG1 = slopeC1*positionG1.x + yIntercept1C12;

double xLowerLimitG2 = (1.0/slopeC2)*( positionG2.y - yInterceptC2 );
double yLowerLimitG2 = slopeC1*positionG2.x + yInterceptC1;

double xUpperLimitG2 = (1.0/slopeC2)*( positionG2.y - yIntercept2C12);
double yUpperLimitG2 = slopeC1*positionG2.x + yIntercept1C12;

if ( positionG1.x >= xLowerLimitG1 && positionG1.x <= xUpperLimitG1 )
{
if ( positionG1.y >= yLowerLimitG1 && positionG1.y <= yUpperLimitG1 )
{
nAtomsPerLayer++;
}
}

if ( positionG2.x >= xLowerLimitG2 && positionG2.x <= xUpperLimitG2 )
{
if ( positionG2.y >= yLowerLimitG2 && positionG2.y <= yUpperLimitG2 )
{
nAtomsPerLayer++;
}
}
}

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//    //account for missing atoms here, which must be checked by hand, unless we write a flag
//    here for atoms that are very close to the boundary
//    nAtomsPerLayer++;
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

return nAtomsPerLayer;
}

void assignGrapheneParams(AtomParams **Graphene, struct vect a1, struct vect a2, struct vect a3,
    struct vect b1, struct vect b2, struct vect b3)
{
    struct vect B1, B2, B12, C1, C2, C12;

    // Define TSB super-lattice vectors
    B1 = 4.0*b1;
    B2 = 4.0*b2;
    B12 = B1 + B2;

    // Define boundaries of TSB super-lattice vectors (purely for calculation)

```

```

C1 = B1 + 2.0*b3;
C2 = B2 + 2.0*b3;
C12 = B12 + 2.0*b3;

double slopeC1 = (C1.y - 2.0*b3.y) / (C1.x - 2.0*b3.x);
double slopeC2 = (C2.y - 2.0*b3.y) / (C2.x - 2.0*b3.x);
double yInterceptC1 = C1.y - slopeC1*C1.x;
double yInterceptC2 = C2.y - slopeC2*C2.x;

double yIntercept1C12 = C12.y - slopeC1*C12.x;
double yIntercept2C12 = C12.y - slopeC2*C12.x;

// Initialize graphene basis vectors, where "1" and "2" denote basis atoms of the graphene
unit cell
struct vect positionG1 = 2.0*b3;
struct vect positionG2 = 2.0*b3;

// Initialize a sufficiently large grid to scan for atoms inside the boundaries
int maxUnitCells1 = 500;
int maxUnitCells2 = 500;

//assign appropriate graphene atom positions and atom indices to class AtomParams
int atomCount = 0;
for (int i=-maxUnitCells1; i<maxUnitCells1; i++)
{
    for (int j=-maxUnitCells2; j<maxUnitCells2; j++)
    {
        positionG1 = i*a1 + j*a2 + a3;
        positionG2 = i*a1 + j*a2 + 2.0*a3;

        double xLowerLimitG1 = (1.0/slopeC2)*( positionG1.y - yInterceptC2 );
        double yLowerLimitG1 = slopeC1*positionG1.x + yInterceptC1;

        double xUpperLimitG1 = (1.0/slopeC2)*( positionG1.y - yIntercept2C12);
        double yUpperLimitG1 = slopeC1*positionG1.x + yIntercept1C12;

        double xLowerLimitG2 = (1.0/slopeC2)*( positionG2.y - yInterceptC2 );
        double yLowerLimitG2 = slopeC1*positionG2.x + yInterceptC1;

        double xUpperLimitG2 = (1.0/slopeC2)*( positionG2.y - yIntercept2C12);
        double yUpperLimitG2 = slopeC1*positionG2.x + yIntercept1C12;

        if ( positionG1.x >= xLowerLimitG1 && positionG1.x <= xUpperLimitG1 )
        {
            if ( positionG1.y >= yLowerLimitG1 && positionG1.y <= yUpperLimitG1 )
            {
                Graphene[atomCount] = new AtomParams(positionG1, atomCount+1);
                atomCount++;
            }
        }

        if ( positionG2.x >= xLowerLimitG2 && positionG2.x <= xUpperLimitG2 )
        {

```

```

    if ( positionG2.y >= yLowerLimitG2 && positionG2.y <= yUpperLimitG2 )
    {
        Graphene[atomCount] = new AtomParams(positionG2, atomCount+1);
        atomCount++;
    }
}

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//    //account for missing atoms here
//    struct vect missingAtomPosition;
//    missingAtomPosition.x = -2.507;
//    missingAtomPosition.y = 46.151;
//    missingAtomPosition.z = 0.000;
//
//    Graphene[atomCount] = new AtomParams(missingAtomPosition, atomCount+1);
//    atomCount++;
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
}

void writeTOP(AtomParams **Atom, AtomParams **Graphene, int nCarbon, int nQuarter, string guest,
string guestSegName, int nAtomsPerTSB, int nAtomsPerGuest, int nAtomsPerLayer)
{
    // Create output TOP file
    stringstream outStream;
    outStream << "tsb35-c" << nCarbon << "_0" << nQuarter << "-quarter_with-" << guest <<
    "_TOP.top";
    ofstream OUT(outStream.str().c_str());
    OUT << setiosflags(ios::fixed) << setprecision(3);

    cout << "TOP file to generate: " << outStream.str() << endl;

    // topology file header
    OUT << "27 1" << endl;
    OUT << endl;
    OUT << "MASS      1 HGA2      1.00800 H ! nonpolar, aliphatic H" << endl;
    OUT << "MASS      2 HGA3      1.00800 H ! nonpolar, aliphatic H" << endl;
    OUT << "MASS      3 HGR61  1.00800 H ! aromatic H" << endl;
    OUT << "MASS      4 HGA4      1.00800 H ! alkene H " << endl;
    OUT << "MASS      5 CG2R61A  12.01100 C ! aromatic C (see notes for convention)" << endl;
    OUT << "MASS      6 CG2R61B  12.01100 C ! aromatic C (see notes for convention)" << endl;
    OUT << "MASS      7 CG2R61C  12.01100 C ! aromatic C (see notes for convention)" << endl;
    OUT << "MASS      8 CG2R61D  12.01100 C ! aromatic C (see notes for convention) " << endl;
    OUT << "MASS      9 CG321    12.01100 C ! aliphatic sp3 C for CH2" << endl;
    OUT << "MASS     10 CG331    12.01100 C ! aliphatic sp3 C for CH3" << endl;
    OUT << "MASS     11 CG2DC1A  12.01100 C ! alkene C (adjacent to central ring)" << endl;
    OUT << "MASS     12 CG2DC1B  12.01100 C ! alkene C (adjacent to central ring)" << endl;
    OUT << "MASS     13 CG2DC1C  12.01100 C ! alkene C (adjacent to central ring)" << endl;
    OUT << "MASS     14 CG2DC1D  12.01100 C ! alkene C (adjacent to peripheral ring)" << endl;
    OUT << "MASS     15 OG301   15.99940 O ! ether O " << endl;
    OUT << "MASS     16 CG2R61  12.01100 C ! aromatic C (graphite)" << endl;
    OUT << endl;
}

```

```

OUT << "AUTO ANGLES DIHE" << endl;
OUT << endl;

    // TSB molecule topology entry
OUT << "RESI TSB          0.00" << endl;
OUT << endl;

string atomName, atomType;
double charge;
for (int i=0; i<nAtomsPerTSB; i++)
{
    atomName = Atom[i]->getAtomName();
    atomType = Atom[i]->getAtomType();
    charge = Atom[i]->getCharge();

    OUT << "ATOM " << left << setw(5) << atomName << right << setw(4) << atomType << right <<
    setw(8) << charge << endl;
}
OUT << endl;

struct bondList *temp;
for (int i=0; i<nAtomsPerTSB; i++)
{
    atomName = Atom[i]->getAtomName();
    temp = Atom[i]->getBond();

    if (temp != NULL)
        OUT << "BOND ";
    while (temp !=NULL)
    {
        OUT << left << setw(4) << atomName << " " << temp->atomName << " ";
        temp = temp->next;
    }
    if (Atom[i]->getBond() != NULL)
        OUT << endl;
}
OUT << endl;
//

    // Guest molecule topology entry
OUT << "RESI " << guestSegName << "          0.00" << endl;
OUT << endl;

int nAtomsTotal = nAtomsPerTSB + nAtomsPerGuest;
for (int i=nAtomsPerTSB; i<nAtomsTotal; i++)
{
    atomName = Atom[i]->getAtomName();
    atomType = Atom[i]->getAtomType();
    charge = Atom[i]->getCharge();

    OUT << "ATOM " << left << setw(5) << atomName << right << setw(4) << atomType << right <<
    setw(8) << charge << endl;
}

```

```

OUT << endl;

for (int i=0; i<nAtomsPerGuest; i++)
{
    atomName = Atom[i]->getAtomName();
    temp = Atom[i]->getBond();

    if (temp != NULL)
        OUT << "BOND ";
    while (temp !=NULL)
    {
        OUT << left << setw(4) << atomName << " " << temp->atomName << " ";
        temp = temp->next;
    }
    if (Atom[i]->getBond() != NULL)
        OUT << endl;
}
OUT << endl;
//

// Graphene layer topology entry
OUT << "RESI GRAP          0.00" << endl;
OUT << endl;

for (int i=0; i<nAtomsPerLayer; i++)
{
    // having trouble with object-orientated assignment of atomName in the header Atom.h, so
    // I moved the if loop into this script
    stringstream ss;

    if (i < 1000) ss << "A" << i;
    else if (i < 2000) ss << "B" << (i-1000);
    else if (i < 3000) ss << "C" << (i-2000);
    else if (i < 4000) ss << "D" << (i-3000);
    else if (i < 5000) ss << "E" << (i-4000);
    else if (i < 6000) ss << "F" << (i-5000);
    else if (i < 7000) ss << "G" << (i-6000);
    else if (i < 8000) ss << "H" << (i-7000);
    else if (i < 9000) ss << "I" << (i-8000);
    else if (i < 10000) ss << "J" << (i-9000);
    else if (i < 11000) ss << "K" << (i-10000);
    else if (i < 12000) ss << "L" << (i-11000);
    else if (i < 13000) ss << "M" << (i-12000);
    else if (i < 14000) ss << "N" << (i-13000);
    else if (i < 15000) ss << "O" << (i-14000);
    else if (i < 16000) ss << "P" << (i-15000);
    else if (i < 17000) ss << "Q" << (i-16000);
    else if (i < 18000) ss << "R" << (i-17000);
    else if (i < 19000) ss << "S" << (i-18000);
    else if (i < 20000) ss << "T" << (i-19000);
    else if (i < 21000) ss << "U" << (i-20000);

    atomName=ss.str();
}

```

```

    OUT << "ATOM " << left << setw(5) << atomName << right << setw(4) << "CG2R61" << right <<
    setw(8) << 0.000 << endl;
}
OUT << endl;

for (int i=0; i<nAtomsPerLayer; i++)
{
    stringstream ss;

    if (i < 1000) ss << "A" << i;
    else if (i < 2000) ss << "B" << (i-1000);
    else if (i < 3000) ss << "C" << (i-2000);
    else if (i < 4000) ss << "D" << (i-3000);
    else if (i < 5000) ss << "E" << (i-4000);
    else if (i < 6000) ss << "F" << (i-5000);
    else if (i < 7000) ss << "G" << (i-6000);
    else if (i < 8000) ss << "H" << (i-7000);
    else if (i < 9000) ss << "I" << (i-8000);
    else if (i < 10000) ss << "J" << (i-9000);
    else if (i < 11000) ss << "K" << (i-10000);
    else if (i < 12000) ss << "L" << (i-11000);
    else if (i < 13000) ss << "M" << (i-12000);
    else if (i < 14000) ss << "N" << (i-13000);
    else if (i < 15000) ss << "O" << (i-14000);
    else if (i < 16000) ss << "P" << (i-15000);
    else if (i < 17000) ss << "Q" << (i-16000);
    else if (i < 18000) ss << "R" << (i-17000);
    else if (i < 19000) ss << "S" << (i-18000);
    else if (i < 20000) ss << "T" << (i-19000);
    else if (i < 21000) ss << "U" << (i-20000);

    atomName=ss.str();

    temp = Graphene[i]->getBond();

    if (temp != NULL)
        OUT << "BOND ";
    while (temp !=NULL)
    {
        OUT << left << setw(4) << atomName << " " << temp->atomName << " ";
        temp = temp->next;
    }
    if (Graphene[i]->getBond() != NULL)
        OUT << endl;
}
OUT << endl;
//
}

int main(int argc, char *argv[])

```

```

{
    ////////////////
    // USER INPUT //
    ////////////////
    if (argc != 6) {
        cout << "Must have 5 command line arguments (nCarbon, nQuarter, guest, guestSegName,
        nAtomsPerGuest)!" << endl;
    }

    int nCarbon = atoi(argv[1]);
    int nQuarter = atoi(argv[2]);
    string guest = argv[3];
    string guestSegName = argv[4];
    int nAtomsPerGuest = atoi(argv[5]);

    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 + nAtomsPerGuest;

// 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;

//////////
// Determine bonding order for the TSB molecule //
//////////
struct vect position1, position2, distance;
for (int i=0; i<nAtomsPerTSB; i++)
{
    for (int j=i+1; j<nAtomsPerTSB; j++)
    {
        position1 = Atom[i]->getPosition();
        position2 = Atom[j]->getPosition();
        distance = position2 - position1;

        if (position1.Norm(distance) < 1.6)

```

```

    {
        Atom[i]->addBond(Atom[j]->getAtomName());
    }
}

cout << "Bonding order for TSB done." << endl;
cout << endl;

//////////
// Read guest input files (from Gaussian09) //
//////////

stringstream inStream2;
inStream2 << "../cpp-input/" << guest << "_cpp-input.dat";
ifstream in2(inStream2.str().c_str());

cout << "Guest input file to be read: " << inStream2.str() << endl;

for (int i=nAtomsPerTSB; i<nAtomsTotal; i++)
{
    in2 >> tempAtomName >> tempAtomType >> tempPosition.x >> tempPosition.y >> tempPosition.z >>
        tempElementType >> tempCharge;
    Atom[i] = new AtomParams(tempAtomName, tempAtomType, tempPosition, tempElementType,
        tempCharge);
}

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

//////////
// Determine bonding order for the guest molecule //
//////////

for (int i=nAtomsPerTSB; i<nAtomsPerGuest; i++)
{
    for (int j=i+1; j<nAtomsTotal; j++)
    {
        position1 = Atom[i]->getPosition();
        position2 = Atom[j]->getPosition();
        distance = position2 - position1;

        if (position1.Norm(distance) < 1.6)
        {
            Atom[i]->addBond(Atom[j]->getAtomName());
        }
    }
}

cout << "Bonding order for " << guest << " done." << 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);

// Use boundaries defined by TSB lattice vectors to calculate atom per graphene alyer
int nAtomsPerLayer = countGrapheneAtoms(a1, a2, a3, b1, b2, b3);
// Create class for graphene atoms
AtomParams *Graphene[nAtomsPerLayer];

cout << "Graphene parameters read and stored in class AtomParams/Graphene." << endl;

// Call to function to go back through grid and assign positions and atom names to the
graphene atoms
assignGrapheneParams(Graphene, a1, a2, a3, b1, b2, b3);

////////////////////////////////////
// Determine bonding order for graphene layer //
////////////////////////////////////
struct vect positionG1, positionG2;
for (int i=0; i<nAtomsPerLayer; i++)
{
    stringstream ss;

    if (i < 1000) ss << "A" << i;
    else if (i < 2000) ss << "B" << (i-1000);
    else if (i < 3000) ss << "C" << (i-2000);
    else if (i < 4000) ss << "D" << (i-3000);
    else if (i < 5000) ss << "E" << (i-4000);
    else if (i < 6000) ss << "F" << (i-5000);
    else if (i < 7000) ss << "G" << (i-6000);
    else if (i < 8000) ss << "H" << (i-7000);
    else if (i < 9000) ss << "I" << (i-8000);
    else if (i < 10000) ss << "J" << (i-9000);
    else if (i < 11000) ss << "K" << (i-10000);
    else if (i < 12000) ss << "L" << (i-11000);
    else if (i < 13000) ss << "M" << (i-12000);
    else if (i < 14000) ss << "N" << (i-13000);
    else if (i < 15000) ss << "O" << (i-14000);
    else if (i < 16000) ss << "P" << (i-15000);
    else if (i < 17000) ss << "Q" << (i-16000);

```

```

else if (i < 18000) ss << "R" << (i-17000);
else if (i < 19000) ss << "S" << (i-18000);
else if (i < 20000) ss << "T" << (i-19000);
else if (i < 21000) ss << "U" << (i-20000);

string atomName=ss.str();

for (int j=i+1; j<nAtomsPerLayer; j++)
{
    positionG1 = Graphene[i]->getPosition();
    positionG2 = Graphene[j]->getPosition();
    distance = Graphene[j]->getPosition() - Graphene[i]->getPosition();

    if (positionG1.Norm(distance) < 1.6)
    {
//Graphene[i]->addBond(Graphene[j]->getAtomName());

stringstream ss;

if (j < 1000) ss << "A" << j;
else if (j < 2000) ss << "B" << (j-1000);
else if (j < 3000) ss << "C" << (j-2000);
else if (j < 4000) ss << "D" << (j-3000);
else if (j < 5000) ss << "E" << (j-4000);
else if (j < 6000) ss << "F" << (j-5000);
else if (j < 7000) ss << "G" << (j-6000);
else if (j < 8000) ss << "H" << (j-7000);
else if (j < 9000) ss << "I" << (j-8000);
else if (j < 10000) ss << "J" << (j-9000);
else if (j < 11000) ss << "K" << (j-10000);
else if (j < 12000) ss << "L" << (j-11000);
else if (j < 13000) ss << "M" << (j-12000);
else if (j < 14000) ss << "N" << (j-13000);
else if (j < 15000) ss << "O" << (j-14000);
else if (j < 16000) ss << "P" << (j-15000);
else if (j < 17000) ss << "Q" << (j-16000);
else if (j < 18000) ss << "R" << (j-17000);
else if (j < 19000) ss << "S" << (j-18000);
else if (j < 20000) ss << "T" << (j-19000);
else if (j < 21000) ss << "U" << (j-20000);

string nextAtomName=ss.str();
Graphene[i]->addBond(nextAtomName);
    }
}

cout << "Bonding order for graphene done." << endl;
cout << endl;

writeTOP(Atom, Graphene, nCarbon, nQuarter, guest, guestSegName, nAtomsPerTSB, nAtomsPerGuest,
nAtomsPerLayer);

```

```
return 0;  
}
```