cryst.h

```
//     МГУ им. М.В. Ломоносова. Кафедра кристаллографии и кристаллохимии
//     1 курс, Халин А.Д.
//     Научный руководитель: д.х.н. проф. Еремин Н.Н.
//     licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0)

#include <iostream>
#include <cmath>
#define PI 3.1415926
using namespace std;

float cut(float a)
// Возвращает 0, если число по модулю меньше 0.0001. Используется в умножении матриц
{
	if (abs(a) < 0.0001)
		a = 0;
	return a;
}
class Vector;
class Matrix
{
private:
	float m[3][3];
public:
	Matrix();			// Конструктор по умолчанию создает единичную матрицу
	Matrix(float);
	Matrix(float, float, float, float, float, float, float, float, float);
	Matrix(Matrix &);
	~Matrix();

	float Get(int, int);


	Matrix Transpose();		// Возвращает транспонированную матрицу
	Matrix Invert();		// Возвращает обратную матрицу
	float Determinant();// Возвращает определитель матрицы

	// Перегрузка операторов для матриц
	friend Matrix operator+ (const Matrix &, const Matrix &);
	friend Matrix operator- (const Matrix &, const Matrix &);
	friend Matrix operator* (const Matrix &, float);			// Умножение матрицы на рациональное число
	friend Matrix operator* (const Matrix &, const Matrix &);	// Умножение двух матриц
	friend ostream & operator<< (ostream &, Matrix &);
	friend istream & operator>> (istream &, Matrix &);
	Matrix & operator= (Matrix & A);
```

```
	Vector MultiplyByVector(Vector &);	// Возвращает вектор, являющийся результатом умножения данной матрицы на вектор

};
Matrix::Matrix()
// Создает единичную матрицу
{
	for (int i=0; i<3; i++)
		for (int j=0; j<3; j++)
			if (i==j)
				m[i][j] = 1;
			else
				m[i][j] = 0;
}
Matrix::Matrix(float n)
// Создает матрицу со всеми элементами равными n
{
		for (int i=0; i<3; i++)
			for (int j=0; j<3; j++)
				m[i][j] = n;

}
Matrix::Matrix(Matrix & A)
// Конструктор копирования
{
	for (int i=0; i<3; i++)
		for (int j=0; j<3; j++)
			m[i][j] = A.m[i][j];
}
Matrix::Matrix(float a, float b,float c,float d,float e,float f,float g,float h,float i)
// Создает матрицу из списка элементов
{
	m[0][0] = a;  m[0][1] = b;  m[0][2] = c;
	m[1][0] = d;  m[1][1] = e;  m[1][2] = f;
	m[2][0] = g;  m[2][1] = h;  m[2][2] = i;
}
Matrix::~Matrix()
{}

float Matrix::Get(int i, int j)
// Возвращает элемент матрицы с заданными индексами
{
	return m[i][j];
}
Matrix Matrix::Transpose()
// Возвращает транспонированную матрицу
{
	Matrix C;
```

```
	for (int i=0; i<3; i++)
		for (int j=0; j<3; j++)
			C.m[i][j] = m[j][i];
	return C;
}
float Matrix::Determinant()
// Возвращает определитель матрицы, найденный по формуле Лапласа
{
	float det;
	det =  m[0][0]*m[1][1]*m[2][2] + m[0][2]*m[1][0]*m[2][1] + m[0][1]*m[1][2]*m[2][0]
		- m[0][2]*m[1][1]*m[2][0] - m[0][1]*m[1][0]*m[2][2] - m[0][0]*m[1][2]*m[2][1];
	return det;
}
Matrix Matrix::Invert()
// Возвращает обратную матрицу
// https://en.wikipedia.org/wiki/Invertible_matrix#Inversion_of_3_×_3_matrices
{
	float  a = m[0][0], b = m[0][1], c = m[0][2],
			d = m[1][0], e = m[1][1], f = m[1][2],
			g = m[2][0], h = m[2][1], i = m[2][2];
	Matrix R(0);

	R.m[0][0] = cut(e*i - f*h);
	R.m[0][1] = cut(c*h - b*i);
	R.m[0][2] = cut(b*f - c*e);
	R.m[1][0] = cut(f*g - d*i);
	R.m[1][1] = cut(a*i - c*g);
	R.m[1][2] = cut(c*d - a*f);
	R.m[2][0] = cut(d*h - e*g);
	R.m[2][1] = cut(b*g - a*h);
	R.m[2][2] = cut(a*e - b*d);
	R = R * (1 / Determinant());
	return R;
}
Matrix operator + (const Matrix & A, const Matrix & B)
// Возвращает сумму двух матриц
{
	Matrix C;
	for (int i=0; i<3; i++)
		for (int j=0; j<3; j++)
			C.m[i][j] = A.m[i][j] + B.m[i][j];
	return C;
}
Matrix operator - (const Matrix & A, const Matrix & B)
// Возвращает разность двух матриц
{
	Matrix C;
	for (int i=0; i<3; i++)
```

```
			for (int j=0; j<3; j++)
				C.m[i][j] = A.m[i][j] - B.m[i][j];
	return C;
}
Matrix operator * (const Matrix & A, float n)
// Возвращает произвеление матрицы на число
{
	Matrix C;
	for (int i=0; i<3; i++)
		for (int j=0; j<3; j++)
			C.m[i][j] = A.m[i][j] * n;
	return C;
}
Matrix operator * (const Matrix & A, const Matrix & B)
// Возвращает произведение матрицы A на матрицу B
{
	Matrix C(0);
	for (int i = 0; i<3; i++)
		for (int j=0; j<3; j++)
			for (int k=0; k<3; k++)
				C.m[i][j] = cut(C.m[i][j] + A.m[i][k] * B.m[k][j]);
	return C;
}
Matrix& Matrix::operator= (Matrix & A)
{
	for (int i=0; i<3; i++)
		for (int j=0; j<3; j++)
			m[i][j] = A.m[i][j];
	return *this;
}
ostream & operator<< (ostream & s, Matrix & A)
// Оператор вывода матрицы в поток
{
	for (int i=0; i<3; i++)
	{
		for (int j=0; j<3; j++)
			s << A.m[i][j] << '\t';
		s << endl;
	}
	cout << endl;
	return s;
}
istream & operator>> (istream & s, Matrix & A)
// Оператор ввода матрицы из потока
{
	float f;
	for (int i=0; i<9; i++)
	{
```

```
			s >> f;
			A.m[i/3][i%3] = f;
		}
		return s;
}
Matrix GetRotationMatrix(char axis, float angle)
// Возвращает матрицу поворота в декартовых координатах вокруг оси axis на угол angle, заданный в градусах
{
		angle *= PI/180;
		float sine = cut(sin(angle));
		float cosine = cut(cos(angle));
		switch (axis)
		{
			case 'x':
			{
				Matrix rotation(1,		0,		0,
								0, cosine, -sine,
								0, sine, cosine);
				return rotation;
				break;
			}
			case 'y':
			{
				Matrix rotation(cosine,	0, sine,
								0,		1,		0,
								-sine, 0, cosine);
				return rotation;
				break;
			}
			case 'z':
			{
				Matrix rotation(cosine, -sine, 0,
								sine, cosine, 0,
								0,		0,		1);
				return rotation;
				break;
			}
			default:
			{
				cout << "Некорректные параметры поворота" << endl;
				Matrix rotation;
				return rotation;
			}
		}
}
class Vector {
private:
		float v[3];
```

```
public:
	Vector();
	Vector(float);
	Vector(float, float, float);
	~Vector();
	float Get(int i);
	void Set(int, float);
	friend ostream & operator<< (ostream &, Vector &);
	friend istream & operator>> (istream &, Vector &);
	friend Vector operator-(const Vector &, const Vector &);
	friend Vector operator+(const Vector &, const Vector &);
	friend Vector operator*(const Vector &, float);
	friend Vector Matrix::MultiplyByVector(Vector &);
};
Vector::Vector()
{
	for (int i=0; i<3; i++) v[i] = (i+1);
}
Vector::Vector(float n)
// Создает вектор с одинаковыми элементами n
{
	for (int i=0; i<3; i++) v[i] = n;
}
Vector::Vector(float x, float y, float z)
// Создает вектор из перечисленных элементов
{
	v[0] = x;
	v[1] = y;
	v[2] = z;
}
Vector::~Vector(){}
float Vector::Get(int i)
// Возвращает элемаент вектора с номером i
{
	return v[i];
}
void Vector::Set(int i, float R)
// Присваивает элементу вектора с номером i значение R
{
	if ((i>=0) && (i<3))
		v[i] = R;
	else cout << "ERROR: Индекс элемента вектора должен лежать в пределах от 0 до 2.\ni = " << i << endl;
}
ostream & operator<< (ostream & s, Vector & V)
// Вывод вектора в поток
{
	s << V.v[0] << ' ' << V.v[1] << ' ' << V.v[2] << endl;
	return s;
```

```
}
istream & operator>> (istream & s, Vector & V)
// Ввод вектора из потока
{
	float a,b,c;
	s >> a >> b >> c;
	V.v[0] = a;
	V.v[1] = b;
	V.v[2] = c;
	return s;
}
Vector operator-(const Vector & A, const Vector & B)
// Возвращает разность двух векторов
{
	Vector result;
	for (int i=0; i<3; i++)
		result.v[i] = A.v[i] - B.v[i];
	return result;
}
Vector operator+(const Vector & A, const Vector & B)
// Возвращает сумму двух векторов
{
	Vector result;
	for (int i=0; i<3; i++)
		result.v[i] = A.v[i] + B.v[i];
	return result;
}
Vector operator*(const Vector & A, float r)
// Возвращает вектор, умноженный на число
{
	Vector result;
	for (int i=0; i<3; i++)
		result.v[i] = A.v[i] * r;
	return result;
}
Vector Matrix::MultiplyByVector(Vector & b)
// Возвращает вектор, являющийся результатом умножения матрицы на вектор
{
	Vector c(0);
	for (int i=0; i<3; i++)
		for (int j=0; j<3; j++)
			c.v[i] = cut(c.v[i] + m[i][j] * b.Get(j));
	return c;
}
void GetNewABC(float & A, float & B, float & C, float & ALPHA, float & BETA, float & GAMMA, float a, float b, float c, float alpha, float
beta, float gamma, Matrix & M)
// Вычисление новых параметров ячейки по известным старым и по матрице перехода от старой СК к новой
// [Загальская Ю.Г., Литвинская Г.П. Геометрическая микрокристаллография, М, изд. Московского ун-та, 1976, с. 130]
```

```
{
	// для компактности выражений сразу вычисляем косинусы углов
	alpha = cut(cos(alpha*PI/180));
	beta = cut(cos(beta*PI/180));
	gamma = cut(cos(gamma*PI/180));
	// Находим модули направляющих векторов
	float VECTORS[3];
	for (int i=0; i<3; i++)
	{
		VECTORS[i] = sqrt (abs( pow(a*M.Get(i, 0), 2) + pow(b*M.Get(i, 1), 2) + pow(c*M.Get(i, 2), 2)
			+ 2*a*b*gamma*M.Get(i,0)*M.Get(i,1) + 2*a*c*beta*M.Get(i,0)*M.Get(i,2) + 2*b*c*alpha*M.Get(i,1)*M.Get(i,2)));
	}
	A = VECTORS[0];
	B = VECTORS[1];
	C = VECTORS[2];
	// Находим углы между направляющими векторами
	float ANGLES[3];
	for (int i=0; i<3; i++)
	{
		int j = (i+1)%3;
		int k = (i+2)%3;
		if (j>k){ int t = j; j=k; k=t; }
		float cosine = a*a*M.Get(j,0)*M.Get(k,0) + b*b*M.Get(j,1)*M.Get(k,1) + c*c*M.Get(j,2)*M.Get(k,2)
					+ a*b * (M.Get(k,0)*M.Get(j,1) + M.Get(j,0)*M.Get(k,1)) * gamma
					+ a*c * (M.Get(k,0)*M.Get(j,2) + M.Get(j,0)*M.Get(k,2)) * beta
					+ b*c * (M.Get(k,1)*M.Get(j,2) + M.Get(j,1)*M.Get(k,2)) * alpha;
		cosine/= VECTORS[j]*VECTORS[k];
				ANGLES[i] = 180/PI*acos(cosine);
	}

	ALPHA = ANGLES[0];
	BETA = ANGLES[1];
	GAMMA = ANGLES[2];
}
```

main.cpp

```
//     МГУ им. М.В. Ломоносова. Кафедра кристаллографии и кристаллохимии
//     1 курс, Халин А.Д.
//     Научный руководитель: д.х.н. проф. Еремин Н.Н.
//     licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0)

#include <iostream>
#include <fstream>
#include <string>
//#include <clocale>
#include <Windows.h>
#include "crystal_f.h"
using namespace std;

Matrix I;     // Единичная матрица

void OpenRead(ifstream & f, char * filetype)
// Принимает введенное с клавиатуры имя файла и открывает его для чтения
{
	char fname[30];

	// Пока файл не будет открыт, повторяем попытки
	while (!f.is_open())
	{
		cout << "Введите имя файла " << filetype << endl;	// Вывод приглашения
		cin >> fname;
		f.open(fname);

		if (!f.is_open())
			cout << "Файл не может быть открыт!\n";
	}
}
void OpenWrite(ofstream & f, const char * fname)
// Открывает файл на запись
{

	f.open(fname);

	if (!f.is_open())
		cout << "Файл не может быть открыт!\n";

}

void main() {
	//setlocale(LC_CTYPE, "rus");
```

```
SetConsoleOutputCP(1251);

ifstream fin, ftrans;      // Файлы ввода
ofstream fout;                    // Файл результатов
int iteration = 1;

bool repeat = true;
while (repeat)
{
	OpenRead(fin, "координат исходной структуры");

	char buff[20];
	char iter[2];
	Matrix M;

	// Имя файла вывода вычисляется как "имя структуры" + номер итерации + ".cryst"
	fin >> buff;
	itoa(iteration, iter, 10);
	string path = buff;
	path += iter;
	path += ".cryst";
	OpenWrite(fout, path.c_str());

	// Вывод имени структуры в файл и на экран
	fout << buff << ' ';
	cout << buff << ' ';
	fin >> buff;
	fout << buff << endl;
	cout << buff << endl << endl;

	// Открывем и читаем файл преобразованияй координат
	OpenRead(ftrans, "преобразования");
	ftrans >> buff;

	if (buff[0] == 'm')              // "matrix" --> считываем матрицу перехода из файла
	{
		ftrans >> M;
	} else if (buff[0] = 'r')  // "rotation" --> вычисляем итоговую матрцу как произведение всех матриц поворота
	{
		M = I;
		char axis;
		float angle;

		while (!ftrans.eof())
		{
			ftrans >> axis >> angle;
			cout << "Поворот вокруг оси " << axis << " на " << angle << " градусов" << endl;
			M = M * GetRotationMatrix(axis, angle);
```

```
        }
} else {
        cout << "Некорректный синтаксис в файле преобразования. Обратитесь к руководству пользователя" << endl;
        M = I;
}

cout << "Матрица перехода от старых координат к новым:" << endl;
cout << M;


// Чтение параметров ячейки из файла и вычисление новых с помощью полученной матрицы преобразрвания
float param[6];
float A, B, C, AL, BE, GA;
for (int i=0; i<6; i++)
{
        fin >> buff;
        fin >> param[i];
}
GetNewABC(A,B,C,AL,BE,GA, param[0], param[1], param[2], param[3], param[4], param[5], M);

// Вывод результатов на экран и в файл
fout << "a\t" << A << "\nb\t" << B << "\nc\t" << C
        << "\nalpha\t" << AL << "\nbeta\t" << BE << "\ngamma\t" << GA << endl;
cout << "a\t" << A << "\nb\t" << B << "\nc\t" << C
        << "\nalpha\t" << AL << "\nbeta\t" << BE << "\ngamma\t" << GA << endl;

// Вычисление обратной транспонированной матрицы для преобразования координат
Matrix M_T = M.Invert().Transpose();

// Построковое считывание координат из файла и вычисление новых
Vector v;
while (!fin.eof()){
        fin >> buff;
        if (!fin.eof())
        {
                fin >> v;
                v = M_T.MultiplyByVector(v);
                fout << buff << ' ' << v;
                cout << buff << ' ' << v;
        }
}
cout << "\nФайл результатов: " << path << endl;

ftrans.close();
fin.close();
fout.close();

iteration++;
```

```
            cout << endl;
        }
        system("pause");

}
```