Vector

```
vector::vector(double x, double y, double z, double w){
	v[0] = x;
	v[1] = y;
	v[2] = z;
	v[3] = w;
}

double vector::Length(void) const {
	return sqrt( v[0]*v[0] + v[1]*v[1] + v[2]*v[2] );
}

void vector::Normalize(void){
	if( !IsEqual(Length(), 0) ) {
		double temp = Length();
		v[0] = v[0] / temp;
		v[1] = v[1] / temp;
		v[2] = v[2] / temp;
	}

}

vector &vector::operator+=(const vector &w){
	v[0] = v[0] + w[0];
	v[1] = v[1] + w[1];
	v[2] = v[2] + w[2];

	return *this;
}

vector &vector::operator-=(const vector &w){
	v[0] = v[0] - w[0];
	v[1] = v[1] - w[1];
	v[2] = v[2] - w[2];

	return *this;
}

vector &vector::operator*=(double d){
	v[0] = v[0] * d;
	v[1] = v[1] * d;
	v[2] = v[2] * d;

	return *this;
}

vector &vector::operator/=(double d){
	if ( d != 0){
		v[0] = v[0] / d;
		v[1] = v[1] / d;
		v[2] = v[2] / d;
	}

	return *this;
}
```

Vector

```
vector vector::Proj(const vector &w){
	double tempValue = ( (v[0]*w[0]) + (v[1]*w[1]) + (v[2]*w[2]) ) / ( w[0]*w[0] +
		w[1]*w[1] + w[2]*w[2] );


	return vector(w[0] * tempValue, w[1] * tempValue, w[2] * tempValue);
}

vector vector::Perp(const vector &w){
	double tempValue = ( (v[0]*w[0]) + (v[1]*w[1]) + (v[2]*w[2]) ) / ( w[0]*w[0] +
		w[1]*w[1] + w[2]*w[2] );

	return vector( v[0]-(w[0] * tempValue) , v[1] - (w[1] * tempValue) , v[2] - (w[2]
		* tempValue) );

}

bool IsZero(const vector &v){
	if(IsEqual(v[0],0), IsEqual(v[1],0), IsEqual(v[2],0)){
		return true;
	}

	return false;
}

bool IsEqual(const vector &v, const vector &w){
	if(IsEqual(v[0],w[0]), IsEqual(v[1],w[1]), IsEqual(v[2],w[2])){
		return true;
	}

	return false;
}

bool IsNotEqual(const vector &v, const vector &w){
	return !IsEqual(v, w);
}

double Dot(const vector &v, const vector &w){
	return ( (v[0]*w[0]) + (v[1]*w[1]) + (v[2]*w[2]) );
}

vector Cross(const vector &v, const vector &w){
	return vector( ( (v[1]*w[2])-(v[2]*w[1]) ) , ( (v[2]*w[0])-(v[0]*w[2]) ) , (
		(v[0]*w[1])-(v[1]*w[0]) ) );
}

vector operator+(const vector &v, const vector &w){
	return vector( (v[0]+w[0]) , (v[1]+w[1]) , (v[2]+w[2]) );
}

vector operator-(const vector &v, const vector &w){
	return vector( (v[0]-w[0]) , (v[1]-w[1]) , (v[2]-w[2]) );
}

vector operator*(double d, const vector &v){
	return vector( (d*v[0]) , (d*v[1]) , (d*v[2]) );
}
```

Vector

```
vector operator*(const vector &v, double d){
	return d * v;
}

vector operator/(const vector &v, double d){
	if( !IsEqual(d,0) ){
		return vector( (v[0]/d) , (v[1]/d) , (v[2]/d) );
	}
	return NULL;
}

point operator+(const point &p, const vector &v){
	return point( (p[0]+v[0]) , (p[1]+v[1]) , (p[2]+v[2]) );
}

point operator+(const vector &v, const point &p){
	return p + v;
}

point &operator+=(point &p, const vector &v){
	p[0] = (p[0]+v[0]);
	p[1] = (p[1]+v[1]);
	p[2] = (p[2]+v[2]);

	return p;
}

point operator-(const point &p, const vector &v){
	return point( (p[0]-v[0]) , (p[1]-v[1]) , (p[2]-v[2]) );
}

point &operator-=(point &p, const vector &v){
	p[0] = (p[0]-v[0]);
	p[1] = (p[1]-v[1]);
	p[2] = (p[2]-v[2]);

	return p;
}

vector operator-(const point &p, const point &q){
	return vector( (p[0]-q[0]) , (p[1]-q[1]) , (p[2]-q[2]) );
}

vector operator-(const vector &v){
	return vector(-v[0], -v[1], -v[2]);
}

istream &operator>>(istream &input, vector &v){
	input >> v[0] >> v[1] >> v[2];
	return input;
}

ostream &operator<<(ostream &output, const vector &v){
	output << v[0] << " " << v[1] << " " << v[2];
	return output;
}
```