Vect2

```
using namespace std;

namespace egc{
        vec2 &vec2 :: operator =(const vec2 &vect){
                this -> x = vect.x;
                this -> y = vect.y;
                return *this;

        }

        vec2 vec2 :: operator +(const vec2 &vect) const { // daca e
                vec2 v = vec2(*this);
                v.x = v.x + vect.x;
                v.y = v.y + vect.y;
                return v;
        }

        vec2 &vec2 ::operator +=(const vec2 &vect) {
                this->x += vect.x;
                this->y += vect.y;
                return *this;
        }

        //vec2 operator *(float scalarValue) const;
        vec2 vec2 ::operator *(float scalarValue) const {
                vec2 v = vec2(*this);
                v.x = v.x*scalarValue;
                v.y = v.y*scalarValue;
                return v;

        }

//      vec2 operator -(const vec2& srcVector) const;
        vec2 vec2::operator -(const vec2& srcVector) const {
                vec2 v = vec2(*this);
                v.x = v.x - srcVector.x;
                v.y = v.y - srcVector.y;
                return v;

        }

        //vec2& operator -=(const vec2& srcVector);

        vec2& vec2::operator -=(const vec2& srcVector) {
                this->x -= srcVector.x;
                this->y -= srcVector.y;
                return *this;
        }

        // float length() const;
        float  vec2:: length() const {
                return sqrt(this->x*this->x + this->y*this->y);

        }
```

```
    //vec2& operator -();



    vec2& vec2::operator -() {
        this->x = this->x *(-1);
        this->y = this->y *(-1);
        return *this;
    }

    //vec2& normalize();

    vec2& vec2:: normalize() {
        *this = *this * (1 / (*this).length());
        return *this;
    }

    float dotProduct(const vec2& v1, const vec2& v2) {
        float dotProduct;
        dotProduct = v1.x * v2.x + v1.y * v2.y;
        return dotProduct;

    }



}
```

Vect 4

```
    //vec4& operator =(const vec4 &srcVector);
    vec4& vec4::operator =(const vec4 &srcVector) {
        this->x = srcVector.x;
        this->y = srcVector.y;
        this->z = srcVector.z;
        this->w = srcVector.w;
        return *this;
    }

    //vec4 operator +(const vec4& srcVector) const;
    vec4 vec4::operator +(const vec4& srcVector) const {
        vec4 v = vec4(*this);
        v.x = v.x + srcVector.x;
        v.y = v.y + srcVector.y;
        v.z = v.z + srcVector.z;
        v.w = v.z + srcVector.w;
        return v;
    }

    //vec4& operator +=(const vec4& srcVector);

    vec4& vec4::operator +=(const vec4& srcVector) {
        this->x += srcVector.x;
        this->y += srcVector.y;
        this->z += srcVector.z;
        this->w += srcVector.w;
        return *this;
    }



    //vec4 operator *(float scalarValue) const;
    vec4 vec4::operator *(float scalarValue) const {
        vec4 v = vec4(*this);
        v.x = v.x*scalarValue;
        v.y = v.y*scalarValue;
        v.z = v.z*scalarValue;
        v.w = v.w*scalarValue;
        return v;
    }


    //vec4 operator -(const vec4& srcVector) const;
    vec4 vec4::operator -(const vec4& srcVector) const {
        vec4 v = vec4(*this);
        v.x = v.x - srcVector.x;
        v.y = v.y - srcVector.y;
        v.z = v.z - srcVector.z;
        v.w = v.w - srcVector.w;
        return v;
    }
    //vec4& operator -=(const vec4& srcVector);
    vec4& vec4::operator -=(const vec4& srcVector) {
        this->x -= srcVector.x;
        this->y -= srcVector.y;
        this->z -= srcVector.z;
        this->w -= srcVector.w;
        return *this;
    }
    //vec4& operator -();
    vec4& vec4::operator -() {
        this->x = this->x *(-1);
        this->y = this->y *(-1);
        this->z = this->z *(-1);
        this->w = this->w *(-1);
        return *this;
    }


    //float length() const;
    float vec4::length() const {
        return sqrt(this->x*this->x + this->y*this->y + this->z*this->z + this-
>w*this->w);
    }
    //vec4& normalize();
    vec4& vec4::normalize() {
        *this = *this * (1 / (*this).length());
        return *this;

    }
}
```

Vect 3

```
       //vec3& operator =(const vec3 &srcVector);

       vec3& vec3 ::operator=(const vec3 &srcVector) {
              this->x = srcVector.x;
              this->y = srcVector.y;
              this->z = srcVector.z;
              return *this;
       }

       //vec3 operator +(const vec3& srcVector) const;
       vec3 vec3::operator+(const vec3& srcVector)const {
              vec3 v = vec3(*this);
              v.x = v.x + srcVector.x;
              v.y = v.y + srcVector.y;
              v.z = v.z + srcVector.z;
              return v;

       }
       //vec3& operator +=(const vec3& srcVector);

       vec3& vec3::operator +=(const vec3& srcVector) {
              this->x += srcVector.x;
              this->y += srcVector.y;
              this->z += srcVector.z;
              return *this;
       }
       //vec3 operator *(float scalarValue) const;

       vec3 vec3::operator*(float scalarValue)const {
              vec3 v = vec3(*this);
              v.x = v.x*scalarValue;
              v.y = v.y*scalarValue;
              v.z = v.z*scalarValue;
              return v;

       }
       //vec3 operator -(const vec3& srcVector) const;

       vec3 vec3::operator-(const vec3& srcVector)const {
              vec3 v = vec3(*this);
              v.x = v.x - srcVector.x;
              v.y = v.y - srcVector.y;
              v.z = v.z - srcVector.z;
              return v;


}
       //vec3& operator -=(const vec3& srcVector);
       vec3 &vec3::operator -=(const vec3& srcVector) {
              this->x -= srcVector.x;
              this->y -= srcVector.y;
              this->z -= srcVector.z;
```

```
                return *this;
        }
        //vec3& operator -();
        vec3& vec3::operator-() {
                this->x = this->x *(-1);
                this->y = this->y *(-1);
                this->z = this->z *(-1);
                return *this;
        }
        //float length() const;

        float vec3::length()const {
                return sqrt(this->x*this->x + this->y*this->y + this->z*this->z);
        }
        //vec3& normalize();
        vec3& vec3::normalize() {
                *this = *this * (1 / (*this).length());
                return *this;
        }
        //float dotProduct(const vec3& v1, const vec3& v2);
        float dotProduct(const vec3& v1, const vec3 &v2) {
                return v1.x * v2.x + v1.y * v2.y+ v1.z * v2.z;

        }

        //vec3 crossProduct(const vec3& v1, const vec3& v2);
        vec3 crossProduct(const vec3 &v1, const vec3 &v2) {
                vec3 m;
                m.x = (v1.y*v2.z) - (v1.z*v2.y);
                m.y = (v1.z*v2.x) - (v1.x*v2.z);
                m.z = (v1.x*v2.y) - (v1.y*v2.x);
                return m;

        }

}
```

Mat 3

```
namespace egc {
        // mat3& operator =(const mat3& srcMatrix);

        mat3& mat3:: operator=(const mat3& srcMatrix) {
                for (int i = 0; i < 9; i++)
                        this->matrixData[i] =  srcMatrix.matrixData[i];
                return *this;
        }
        //mat3 operator *(float scalarValue) const;

        mat3 mat3::operator *(float scalarValue) const {
                mat3 m = mat3(*this);
                for (int i = 0; i < 9; i++) {
                        m.matrixData[i] = m.matrixData[i] * scalarValue;
                        }
                return m;
```

```
    }
    //mat3 operator *(const mat3& srcMatrix) const;


        mat3 mat3 :: operator *(const mat3& srcMatrix) const {
            mat3 m = mat3(*this);
            for (int i = 0; i <= 2; i++) {
                for (int j = 0; j <= 2; j++) {
                    float sum = 0;
                    for (int k = 0; k <= 2; k++) {
                        sum = sum + this->at(i, k)*srcMatrix.at(k, j);
                    }
                    m.matrixData[3 * j + i] = sum;
                }
            }
            return m;
        }



    //vec3 operator *(const vec3& srcVector) const;
    vec3 mat3::operator *(const vec3& srcVector) const {
        vec3 v= vec3();


        v.x = this->matrixData[0] * srcVector.x + this->matrixData[3] * srcVector.y
+ this->matrixData[6] * srcVector.z;
        v.y = this->matrixData[1] * srcVector.x + this->matrixData[4] * srcVector.y
+ this->matrixData[7] * srcVector.z;
        v.z = this->matrixData[2] * srcVector.x + this->matrixData[5] * srcVector.y
+ this->matrixData[8] * srcVector.z;
        return v;

    }
    //mat3 operator +(const mat3& srcMatrix) const;
    mat3 mat3::operator +(const mat3& srcMatrix) const {
        mat3 m;
        for (int i = 0; i < 9; i++)
            m.matrixData[i] = this->matrixData[i] + srcMatrix.matrixData[i];
        return m;
    }




    //float& at(int i, int j);
    float& mat3::at(int i, int j) {
        return this->matrixData[3 * j + i];
    }
    //const float& at(int i, int j) const;
    const float& mat3::at(int i, int j) const {
        return this->matrixData[3 * j + i];
    }

    //float determinant() const;
    float mat3::determinant() const {
        float det = 0, sum1 = 0, sum2 = 0;
```

```
              sum1 = this->at(0, 0)*this->at(1, 1)*this->at(2, 2) + this->at(0, 1)*this-
>at(1, 2)*this->at(2, 0) + this->at(0, 2)*this->at(1, 0)*this->at(2, 1);
              sum2 = this->at(0, 2)*this->at(1, 1)*this->at(2, 0) + this->at(0, 0)*this-
>at(1, 2)*this->at(2, 1) + this->at(0, 1)*this->at(1, 0)*this->at(2, 2);
              det = sum1 - sum2;
              return det;
       }



       //mat3 inverse() const;
       mat3 mat3::inverse() const {
              mat3 mat;
              float demp = 0;
              mat.at(0, 0) = this->at(1, 1)*this->at(2, 2) - this->at(1, 2)*this->at(2,
1);
              mat.at(1, 0) = (-1)*(this->at(1, 0)*this->at(2, 2) - this->at(1, 2)*this-
>at(2, 0));
              mat.at(2, 0) = this->at(1, 0)*this->at(2, 1) - this->at(1, 1)*this->at(2,
0);

              mat.at(0, 1) = (-1)*(this->at(0, 1)*this->at(2, 2) - this->at(0, 2)*this-
>at(2, 1));
              mat.at(1, 1) = this->at(0, 0)*this->at(2, 2) - this->at(0, 2)*this->at(2,
0);
              mat.at(2, 1) = (-1)*(this->at(0, 0)*this->at(2, 1) - this->at(0, 1)*this-
>at(2, 0));

              mat.at(0, 2) = (this->at(0, 1)*this->at(1, 2) - this->at(0, 2)*this->at(1,
1));
              mat.at(1, 2) = (-1)*(this->at(0, 0)*this->at(1, 2) - this->at(0, 2)*this-
>at(1, 0));
              mat.at(2, 2) = this->at(0, 0)*this->at(1, 1) - this->at(0, 1)*this->at(1,
0);

              float det = 1 / this->determinant();

              mat = mat * det;
              return mat;
       }

       //mat3 transpose() const;
       mat3 mat3::transpose() const {
              mat3 m;
              m.at(0, 0) = this->at(0, 0);
              m.at(0, 1) = this->at(1, 0);
              m.at(0, 2) = this->at(2, 0);

              m.at(1, 0) = this->at(0, 1);
              m.at(1, 1) = this->at(1, 1);
              m.at(1, 2) = this->at(2, 1);

              m.at(2, 0) = this->at(0, 2);
              m.at(2, 1) = this->at(1, 2);
              m.at(2, 2) = this->at(2, 2);

              return m;
       }
```

MAT 4

```
       //mat4& operator =(const mat4& srcMatrix);

       mat4 &mat4::operator = (const mat4& srcMatrix) {
              for (int i = 0; i < 16; i++)
                     this->matrixData[i] = srcMatrix.matrixData[i];
              return *this;

       }
       //mat4 operator *(float scalarValue) const;
       mat4 mat4::operator *(float scalarValue) const {
              mat4 m = mat4(*this);
              for (int i = 0; i < 16; i++)
                     m.matrixData[i] = scalarValue * m.matrixData[i];
              return m;
       }


       //mat4 operator *(const mat4& srcMatrix) const;

       mat4 mat4::operator*(const mat4& srcMatrix) const {
              mat4 m = mat4(*this);
              for (int i = 0; i <= 3; i++) {
                     for (int j = 0; j <= 3; j++) {
                            float sum = 0;
                            for (int k = 0; k <= 3; k++) {
                                   sum = sum + this->at(i, k)*srcMatrix.at(k, j);
                            }
                            m.matrixData[4 * j + i] = sum;
                     }
              }
              return m;
       }

       //vec4 operator *(const vec4& srcVector) const;
       vec4 mat4::operator *(const vec4& srcVector) const {
              vec4 v;
              v.x = this->matrixData[0] * srcVector.x + this->matrixData[4] * srcVector.y
+ this->matrixData[8] * srcVector.z + this->matrixData[12] * srcVector.w;
              v.y = this->matrixData[1] * srcVector.x + this->matrixData[5] * srcVector.y
+ this->matrixData[9] * srcVector.z + this->matrixData[13] * srcVector.w;
              v.z = this->matrixData[2] * srcVector.x + this->matrixData[6] * srcVector.y
+ this->matrixData[10] * srcVector.z + this->matrixData[14] * srcVector.w;
              v.w = this->matrixData[3] * srcVector.x + this->matrixData[7] * srcVector.y
+ this->matrixData[11] * srcVector.z + this->matrixData[15] * srcVector.w;

              return v;
       }

       //mat4 operator +(const mat4& srcMatrix) const;
       mat4 mat4::operator +(const mat4& srcMatrix) const {
              mat4 m;
              for (int i = 0; i < 16; i++)
                     m.matrixData[i] = this->matrixData[i] + srcMatrix.matrixData[i];
              return m;
       }
       //float& at(int i, int j);
       float& mat4::at(int i, int j) {
              return this->matrixData[4 * j + i];
       }

       //const float& at(int i, int j) const;
   const float& mat4::at(int i, int j) const {
       return this->matrixData[4 * j + i];
    }

//float determinant() const;

   float mat4::determinant() const {
          float det = 0, sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;

          sum1 = this->matrixData[0] * (this->matrixData[5] * this->matrixData[10] *
this->matrixData[15] + this->matrixData[9] * this->matrixData[14] * this->matrixData[7] +
this->matrixData[13] * this->matrixData[6] * this->matrixData[11] - this->matrixData[7] *
this->matrixData[10] * this->matrixData[13] - this->matrixData[11] * this->matrixData[14]
* this->matrixData[5] - this->matrixData[9] * this->matrixData[6] * this-
>matrixData[15]);
          sum2 = this->matrixData[1] * (this->matrixData[4] * this->matrixData[10] *
this->matrixData[15] + this->matrixData[8] * this->matrixData[14] * this->matrixData[7] +
this->matrixData[12] * this->matrixData[6] * this->matrixData[11] - this->matrixData[7] *
this->matrixData[10] * this->matrixData[12] - this->matrixData[11] * this->matrixData[14]
* this->matrixData[4] - this->matrixData[8] * this->matrixData[6] * this-
>matrixData[15]);
          sum3 = this->matrixData[2] * (this->matrixData[4] * this->matrixData[9] * this-
>matrixData[15] + this->matrixData[8] * this->matrixData[13] * this->matrixData[7] +
this->matrixData[12] * this->matrixData[5] * this->matrixData[11] - this->matrixData[7] *
this->matrixData[9] * this->matrixData[12] - this->matrixData[11] * this->matrixData[13]
* this->matrixData[4] - this->matrixData[9] * this->matrixData[5] * this-
>matrixData[15]);
          sum4 = this->matrixData[3] * (this->matrixData[4] * this->matrixData[9] * this-
>matrixData[14] + this->matrixData[8] * this->matrixData[13] * this->matrixData[6] +
this->matrixData[12] * this->matrixData[5] * this->matrixData[10] - this->matrixData[6] *
this->matrixData[9] * this->matrixData[12] - this->matrixData[10] * this->matrixData[13]
* this->matrixData[4] - this->matrixData[8] * this->matrixData[5] * this-
>matrixData[14]);

          det = sum1 - sum2 + sum3 - sum4;

          return det;
   }
//mat4 inverse() const;
   mat4 mat4::inverse() const {
          mat4 mat = mat4();
          float demp = 0;


          for (int i = 0; i < 16; i++)
          {
                 for (int j = 0; j < 16; j++)
                 {
                        mat.at(j, i) = (this->at((i + 1) % 100, (+1) % 100)*
                               this->at((i + 2) % 100, (j + 2) % 100) - (this->at((i + 1)
% 100, (j + 2) % 100) *
```

```
                                      this->at((i + 2) % 00, (j + 1) % 100)) / this-
>determinant());
                 }


                 return mat;
          }
   }
   //mat4 transpose() const;
   mat4 mat4::transpose() const {
          mat4 mat = mat4();
          mat.at(0, 0) = this->at(0, 0);
          mat.at(0, 1) = this->at(1, 0);
          mat.at(0, 2) = this->at(2, 0);
          mat.at(0, 3) = this->at(3, 0);

          mat.at(1, 0) = this->at(0, 1);
          mat.at(1, 1) = this->at(1, 1);
          mat.at(1, 2) = this->at(2, 1);
          mat.at(1, 3) = this->at(3, 1);

          mat.at(2, 0) = this->at(0, 2);
          mat.at(2, 1) = this->at(1, 2);
          mat.at(2, 2) = this->at(2, 2);
          mat.at(2, 3) = this->at(3, 2);

          mat.at(3, 0) = this->at(0, 3);
          mat.at(3, 1) = this->at(1, 3);
          mat.at(3, 2) = this->at(2, 3);
          mat.at(3, 3) = this->at(3, 3);

          return mat;
   }
}
```

TRANSLATII

```
namespace egc {
       //transformation matrices in 2D
       //mat3 translate(const vec2 translateArray);
       mat3 translate(const vec2 translateArray) {
              mat3 v;
              v.matrixData[0] = 1;
              v.matrixData[1] = 0;
              v.matrixData[2] = 0;
              v.matrixData[3] = 0;
              v.matrixData[4] = 1;
              v.matrixData[5] = 0;
              v.matrixData[6] = translateArray.x;
              v.matrixData[7] = translateArray.y;
              v.matrixData[8] = 1;
              return v;
       }
       //mat3 translate(float tx, float ty);
       mat3 translate(float tx, float ty) {
              mat3 v;
              v.matrixData[0] = 1;
              v.matrixData[1] = 0;
              v.matrixData[2] = 0;
              v.matrixData[3] = 0;
              v.matrixData[4] = 1;
              v.matrixData[5] = 0;
              v.matrixData[6] = tx;
              v.matrixData[7] = ty;
              v.matrixData[8] = 1;
              return v;
       }


       //mat3 scale(const vec2 scaleArray);
       mat3 scale(const vec2 scaleArray) {
              mat3 v;
              v.matrixData[0] = scaleArray.x;
              v.matrixData[1] = 0;
              v.matrixData[2] = 0;
              v.matrixData[3] = 0;
              v.matrixData[4] = scaleArray.y;
              v.matrixData[5] = 0;
              v.matrixData[6] = 0;
              v.matrixData[7] = 0;
              v.matrixData[8] = 1;
              return v;
       }
       //mat3 scale(float sx, float sy);
       mat3 scale(float sx, float sy) {
              mat3 v;
              v.matrixData[0] = sx;
              v.matrixData[1] = 0;
              v.matrixData[2] = 0;
              v.matrixData[3] = 0;
              v.matrixData[4] = sy;
              v.matrixData[5] = 0;
              v.matrixData[6] = 0;
              v.matrixData[7] = 0;
              v.matrixData[8] = 1;
              return v;
       }
       //mat3 rotate(float angle);
       mat3 rotate(float angle) {
              mat3 v;
              v.matrixData[0] = cos(angle*(PI / 180));
              v.matrixData[1] = sin(angle*(PI / 180));
              v.matrixData[2] = 0;
              v.matrixData[3] = -sin(angle*(PI / 180));
              v.matrixData[4] = cos(angle*(PI / 180));
              v.matrixData[5] = 0;
              v.matrixData[6] = 0;
              v.matrixData[7] = 0;
              v.matrixData[8] = 1;
              return v;

       }
       ////transformation matrices in 3D
```

```
    //mat4 translate(const vec3 translateArray);
    mat4 translate(const vec3 translateArray) {
        mat4 v;
        v.matrixData[0] = 1;
        v.matrixData[1] = 0;
        v.matrixData[2] = 0;
        v.matrixData[3] = 0;
        v.matrixData[4] = 0;
        v.matrixData[5] = 1;
        v.matrixData[6] = 0;
        v.matrixData[7] = 0;
        v.matrixData[8] = 0;
        v.matrixData[9] = 0;
        v.matrixData[10] = 1;
        v.matrixData[11] = 0;
        v.matrixData[12] = translateArray.x;
        v.matrixData[13] = translateArray.y;
        v.matrixData[14] = translateArray.z;
        v.matrixData[15] = 1;

        return v;

    }

    //mat4 translate(float tx, float ty, float tz);
    mat4 translate(float tx, float ty, float tz) {
    mat4 v;
    v.matrixData[0] = 1;
    v.matrixData[1] = 0;
    v.matrixData[2] = 0;
    v.matrixData[3] = 0;
    v.matrixData[4] = 0;
    v.matrixData[5] = 1;
    v.matrixData[6] = 0;
    v.matrixData[7] = 0;
    v.matrixData[8] = 0;
    v.matrixData[9] = 0;
    v.matrixData[10] = 1;
    v.matrixData[11] = 0;
    v.matrixData[12] = tx;
    v.matrixData[13] = ty;
    v.matrixData[14] = tz;
    v.matrixData[15] = 1;

    return v;
}

    //mat4 scale(const vec3 scaleArray);
    mat4 scale(const vec3 scaleArray) {
        mat4 v;
        v.matrixData[0] = scaleArray.x;
        v.matrixData[1] = 0;
        v.matrixData[2] = 0;
        v.matrixData[3] = 0;
        v.matrixData[4] = 0;
        v.matrixData[5] = scaleArray.y;
        v.matrixData[6] = 0;
        v.matrixData[7] = 0;
```

```
	v.matrixData[8] = 0;
	v.matrixData[9] = 0;
	v.matrixData[10] = scaleArray.z;
	v.matrixData[11] = 0;
	v.matrixData[12] = 0;
	v.matrixData[13] = 0;
	v.matrixData[14] = 0;
	v.matrixData[15] = 1;

	return v;
}
//mat4 scale(float sx, float sy, float sz);
mat4 scale(float sx, float sy, float sz) {
	mat4 v;
	v.matrixData[0] = sx;
	v.matrixData[1] = 0;
	v.matrixData[2] = 0;
	v.matrixData[3] = 0;
	v.matrixData[4] = 0;
	v.matrixData[5] = sy;
	v.matrixData[6] = 0;
	v.matrixData[7] = 0;
	v.matrixData[8] = 0;
	v.matrixData[9] = 0;
	v.matrixData[10] = sz;
	v.matrixData[11] = 0;
	v.matrixData[12] = 0;
	v.matrixData[13] = 0;
	v.matrixData[14] = 0;
	v.matrixData[15] = 1;

	return v;
}

//mat4 rotateZ(float angle);
mat4 rotateZ(float angle) {
	mat4 v;
	v.matrixData[0] = cos(angle*(PI / 180));
	v.matrixData[1] = sin(angle*(PI / 180));
	v.matrixData[2] = 0;
	v.matrixData[3] = 0;
	v.matrixData[4] = -sin(angle*(PI / 180));
	v.matrixData[5] = cos(angle*(PI / 180));
	v.matrixData[6] = 0;
	v.matrixData[7] = 0;
	v.matrixData[8] = 0;
	v.matrixData[9] = 0;
	v.matrixData[10] = 1;
	v.matrixData[11] = 0;
	v.matrixData[12] = 0;
	v.matrixData[13] = 0;
	v.matrixData[14] = 0;
	v.matrixData[15] = 1;

	return v;

}
```

```
		//mat4 rotateX(float angle);
		mat4 rotateX(float angle) {
			mat4 v;
			v.matrixData[0] = 1;
			v.matrixData[1] = 0;
			v.matrixData[2] = 0;
			v.matrixData[3] = 0;
			v.matrixData[4] = 0;
			v.matrixData[5] = cos(angle*(PI / 180));
			v.matrixData[6] = sin(angle*(PI / 180));
			v.matrixData[7] = 0;
			v.matrixData[8] = 0;
			v.matrixData[9] = -sin(angle*(PI / 180));
			v.matrixData[10] = cos(angle*(PI / 180));
			v.matrixData[11] = 0;
			v.matrixData[12] = 0;
			v.matrixData[13] = 0;
			v.matrixData[14] = 0;
			v.matrixData[15] = 1;

			return v;


		}

		//mat4 rotateY(float angle);
		mat4 rotateY(float angle) {
			mat4 v;
			v.matrixData[0] = cos(angle*(PI / 180));
			v.matrixData[1] = 0;
			v.matrixData[2] = -sin(angle*(PI/180));
			v.matrixData[3] = 0;
			v.matrixData[4] = 0;
			v.matrixData[5] = 1;
			v.matrixData[6] = 0;
			v.matrixData[7] = 0;
			v.matrixData[8] = sin(angle*(PI / 180));
			v.matrixData[9] = 0;
			v.matrixData[10] = cos(angle*(PI / 180));
			v.matrixData[11] = 0;
			v.matrixData[12] = 0;
			v.matrixData[13] = 0;
			v.matrixData[14] = 0;
			v.matrixData[15] = 1;

			return v;
		}
	}
```