Quaternion Operations

Overview of the Code:

Files Structure:

- **main.c**: Contains the main function, and displays the results after operations in quaternion like complement, multiplication and dot.
- **3_component_vector**.c -Contains function definitions related to vector operations like addition, cross product, dot product, and scalar multiplication.
- **main.h**: Header file that defines the structures for vectors and quaternions and declares all the functions for vector and quaternion operations.
- **quaternion.c**: Contains the implementation of the functions for quaternion operations.

Vector and Quaternion Structures:

We define two structures: one for 3D vectors and another for quaternions.

- **Vector Structure**:

  Represents a 3-component vector (i, j, k) used in vector operations.

```
typedef struct {
  float i;
  float j;
  float k;
} Vector;
```

- **Quaternion Structure**: Represents a quaternion with a scalar component s and a vector component v.

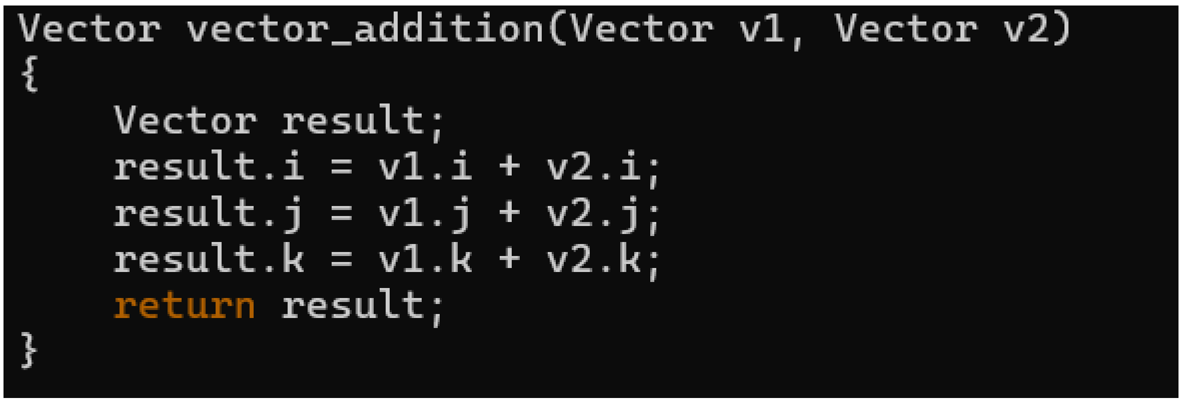
```
typedef struct {
  float s; // Scalar component
  Vector v; // Vector component (i, j, k)
} Quaternion;
```

Summary of Functions:

Vector Operations:

1. **Vector Addition**: Adds two vectors component-wise.

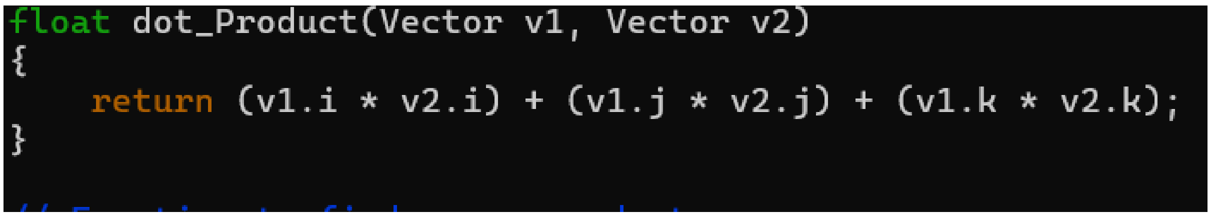
   Vector vector_addition(Vector v1, Vector v2);



```
Vector vector_addition(Vector v1, Vector v2)
{
    Vector result;
    result.i = v1.i + v2.i;
    result.j = v1.j + v2.j;
    result.k = v1.k + v2.k;
    return result;
}
```

2. **Dot Product**: Computes the dot product of two vectors.

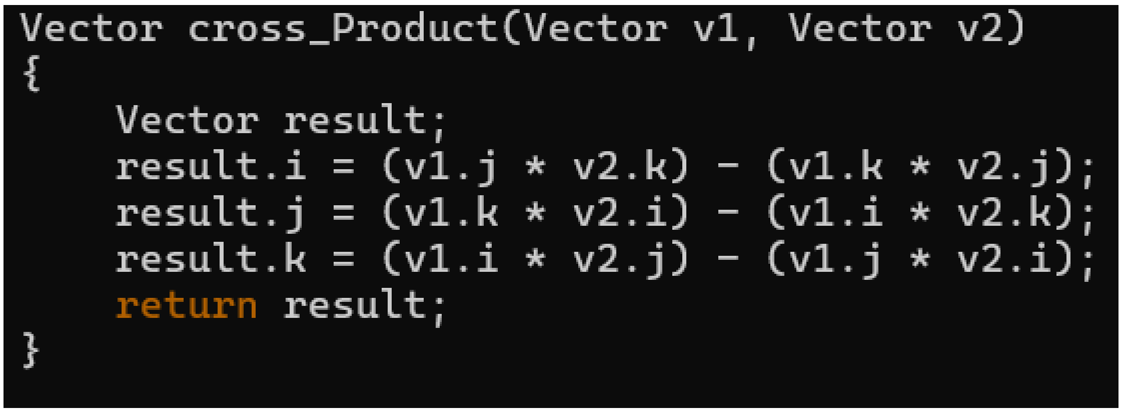
   float dot_Product(Vector v1, Vector v2);



```
float dot_Product(Vector v1, Vector v2)
{
    return (v1.i * v2.i) + (v1.j * v2.j) + (v1.k * v2.k);
}
```

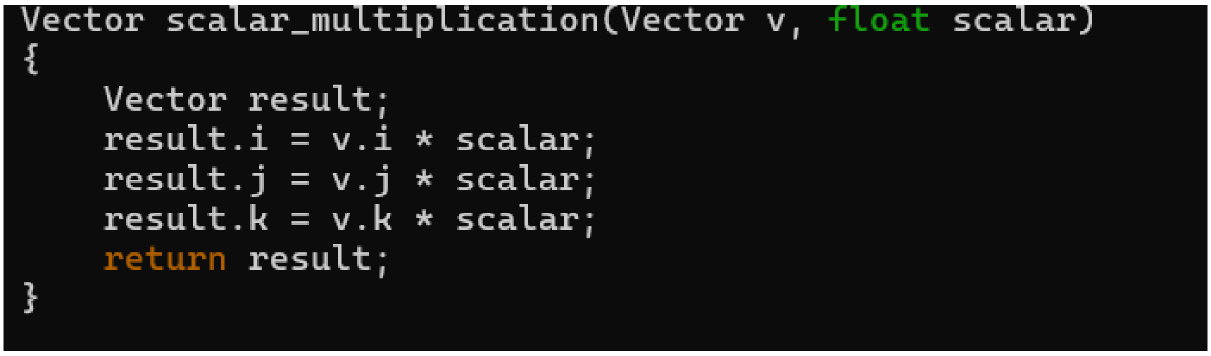
3. **Cross Product**: Computes the cross product of two vectors.

   Vector cross_Product(Vector v1, Vector v2);



```
Vector cross_Product(Vector v1, Vector v2)
{
    Vector result;
    result.i = (v1.j * v2.k) - (v1.k * v2.j);
    result.j = (v1.k * v2.i) - (v1.i * v2.k);
    result.k = (v1.i * v2.j) - (v1.j * v2.i);
    return result;
}
```

4. **Scalar Multiplication**: Multiplies a vector by a scalar.
   Vector scalar_multiplication(Vector v, float scalar);



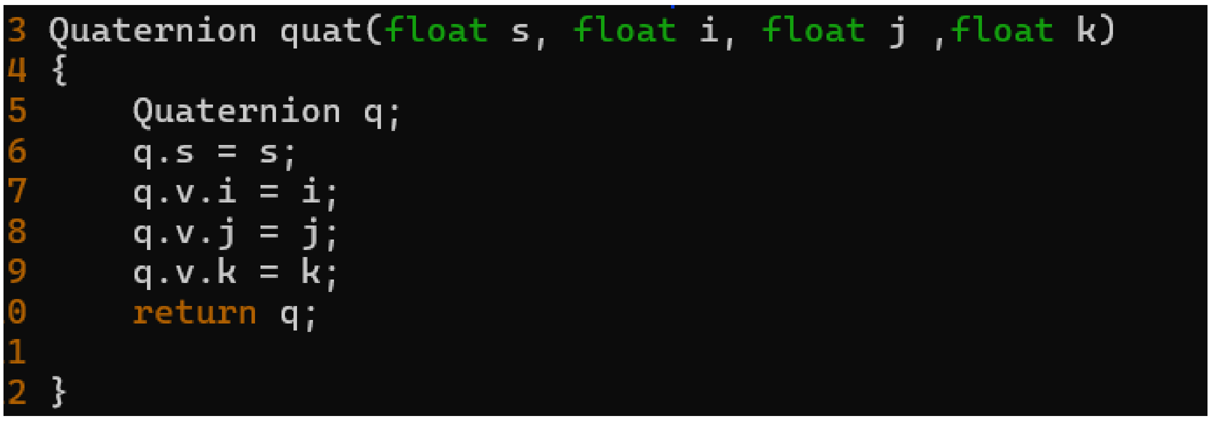
```
Vector scalar_multiplication(Vector v, float scalar)
{
    Vector result;
    result.i = v.i * scalar;
    result.j = v.j * scalar;
    result.k = v.k * scalar;
    return result;
}
```

Quaternion Operations

1. **Quaternion Initialization**: Initializes a quaternion given its scalar and vector components.

Quaternion quat(float s, float i, float j, float k);

By this function we are initializing the quaternion by this function.



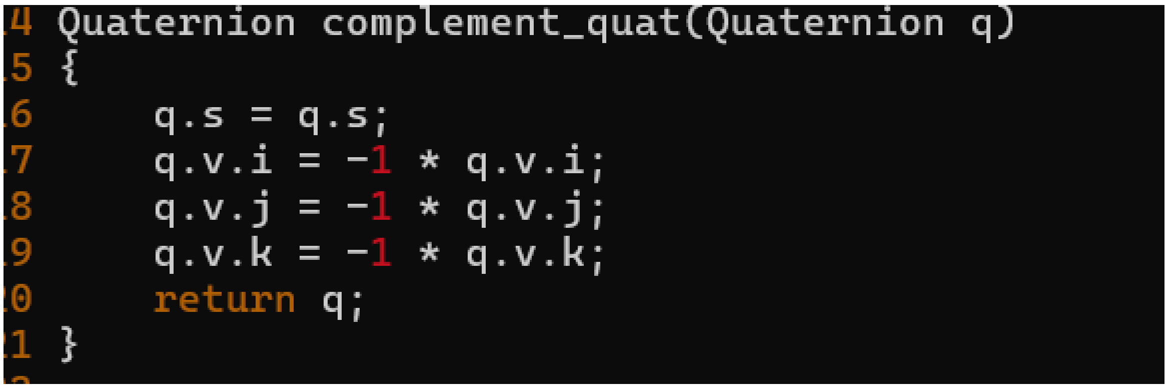
```
Quaternion quat(float s, float i, float j ,float k)
{
    Quaternion q;
    q.s = s;
    q.v.i = i;
    q.v.j = j;
    q.v.k = k;
    return q;

}
```

2. **Complement (Conjugate)**: Computes the conjugate (complement) of a quaternion by negating the vector part.

Quaternion complement_quat(Quaternion q);

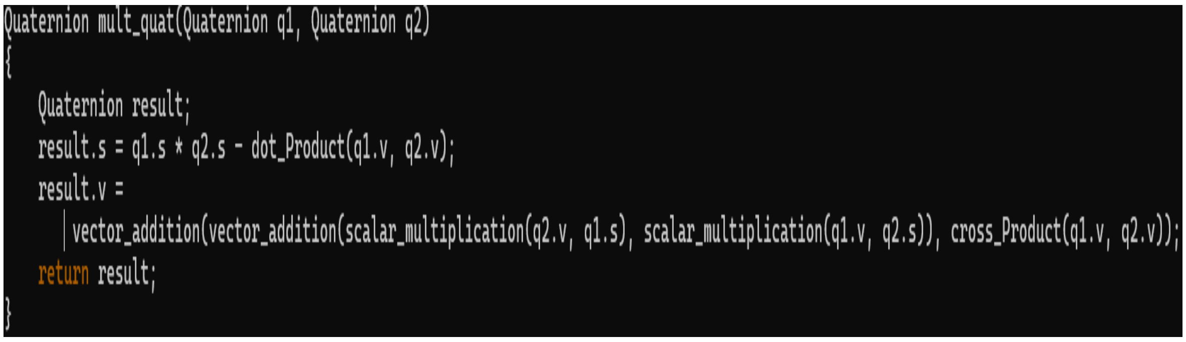
In this we are finding the complement of the quaternion, its nothing but negating the vector part



```
Quaternion complement_quat(Quaternion q)
{
    q.s = q.s;
    q.v.i = -1 * q.v.i;
    q.v.j = -1 * q.v.j;
    q.v.k = -1 * q.v.k;
    return q;
}
```

3. **Quaternion Multiplication**: Multiplies two quaternions using the formula for quaternion multiplication.

Quaternion mult_quat(Quaternion q1, Quaternion q2);



```
Quaternion mult_quat(Quaternion q1, Quaternion q2)
{
    Quaternion result;
    result.s = q1.s * q2.s - dot_Product(q1.v, q2.v);
    result.v =
        vector_addition(vector_addition(scalar_multiplication(q2.v, q1.s), scalar_multiplication(q1.v, q2.s)), cross_Product(q1.v, q2.v));
    return result;
}
```

Suppose:

q1 = 1,2,3,4(s1,i1, j1,k1) and q2 = 0,1,1,1(s2,i2,j2,k2);

result(s3, i3, j3, k3)

Scalar part:

s3 = s1×s2 − i1×i2 − j1×j2 − k1×k2

Vector part:

i3 = s1×i2 + i1×s2 + j1×k2 − k1×j2

j3 = s1×j2 + j1×s2 + k1×i2 − i1×k2

k3 = s1×k2 + k1×s2 + i1×j2 − j1×i2

So, for scalar part the result, we need to multiply scalar part the two quaternions, then subtract with the dot product of the vector part of the two quaternions

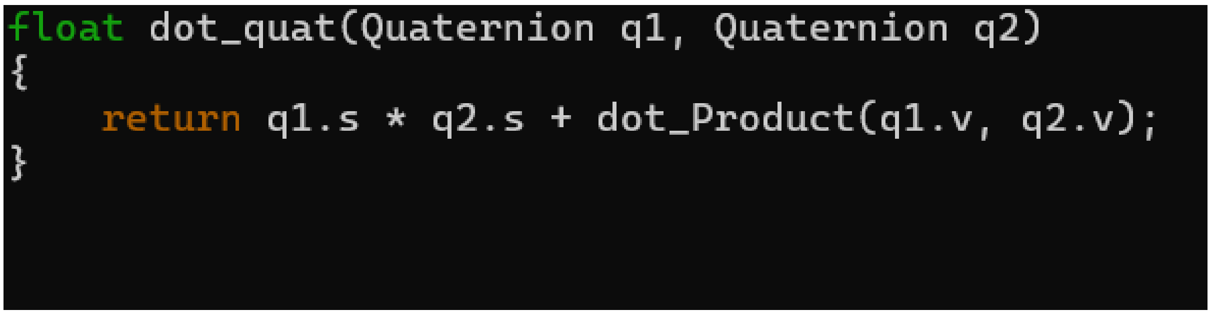
For vector part of the result, first we need to multiple each dimension with the scalar part of each quaternion and add it

i.e. scalar multiplication of (q1.v, q2.s) + scalar multiplication of (q2.v, q1.s).

Then we need to add with the cross product of that particular dimension

4. **Dot Product of Quaternions**: Computes the dot product between two quaternions by combining the dot product of their scalar and vector parts.

float dot_quat(Quaternion q1, Quaternion q2);



```
float dot_quat(Quaternion q1, Quaternion q2)
{
    return q1.s * q2.s + dot_Product(q1.v, q2.v);
}
```