```cpp
#include "kernelutilities.h"
#define _USE_MATH_DEFINES


Vector3f KernelUtilities::gradSpikyKernel(Vector3f r)
{
    // Gradient of Eq(21)
    float rmag = r.abs();
    Vector3f grad;

    if (0 <= rmag && rmag <= h)
    {
        float constant = SPIKY_KERNEL_GRAD_CONSTANT * pow(h - rmag, 2.0) / rmag ;
        grad = constant * r;
    }

    else
    {
        grad = Vector3f::ZERO;
    }

    return grad;
}

float KernelUtilities::polySixKernel(Vector3f r)
{
    // Eq(20)
    float result;
    float rmag = r.abs();

    if (0 <= rmag && rmag <= h)
    {
        float innerPart = pow( (pow(h, 2.0) - r.absSquared()), 3.0 );
        result = POLY_SIX_KERNEL_CONSTANT * innerPart;
    }

    else
    {
        result = 0.0;
    }

    return result;
}

Vector3f KernelUtilities::gradPolySixKernel(Vector3f r)
{
    float rMagSquared = r.absSquared();
    return GRAD_POLY_SIX_KERNEL_CONSTANT * pow( (pow(h, 2.0) - rMagSquared), 2.0) * r;
}

float KernelUtilities::laplacianPolySixKernel(Vector3f r)
{
    float rmag = r.abs();
    float laplacian;

    if (0 <= rmag && rmag <= h)
    {
        float hSquared = pow(h, 2.0);
        float rMagSquared = r.absSquared();
        float firstTerm = LAPLACIAN_POLY_SIX_KERNEL_CONSTANT * (hSquared - rMagSquared);
        float secondTerm = rMagSquared - 0.75 * (hSquared - rMagSquared);
        laplacian = firstTerm * secondTerm;
    }

    else
    {
        laplacian = 0.0f;
    }
```

```
    return laplacian;
}

float KernelUtilities::laplacianViscosityKernel(Vector3f r)
{
    // Laplacian of Eq(22); computed for us in the paper
    float rmag = r.abs();
    float laplacian;

    if (0 <= rmag && rmag <= h)
    {
        laplacian = LAPLACIAN_VISCOSITY_KERNEL_CONSTANT * (h - rmag);
    }

    else
    {
        laplacian = 0.0f;
    }

    return laplacian;
}
```