```cpp
// COS30008, Tutorial 3 + Problem Set 1/2, 2022

#include "Polynomial.h"
#include <iostream>
#include <cmath>

// Default constructor
// Initializes polynomial with degree 0 and all coefficients to 0
Polynomial::Polynomial()
{
    fDegree = 0;

    for (size_t i = 0; i <= MAX_DEGREE; i++)
    {
        fCoeffs[i] = 0.0;
    }
}

// Binary operator* to multiply two polynomials
// Returns a new polynomial with degree = this.degree + aRHS.degree
Polynomial Polynomial::operator*(const Polynomial& aRHS) const
{
    Polynomial result;
    result.fDegree = fDegree + aRHS.fDegree;

    for (size_t i = 0; i <= fDegree; i++)
    {
        for (size_t j = 0; j <= aRHS.fDegree; j++)
        {
            result.fCoeffs[i + j] += fCoeffs[i] * aRHS.fCoeffs[j];
        }
    }

    return result;
}

// Binary operator== to compare two polynomials
// Returns true if polynomials are structurally equivalent
bool Polynomial::operator==(const Polynomial& aRHS) const
{
    if (fDegree != aRHS.fDegree)
        return false;

    for (size_t i = 0; i <= fDegree; i++)
    {
        if (fCoeffs[i] != aRHS.fCoeffs[i])
            return false;
    }
    return true;
}

// Input operator for polynomials (highest to lowest)
std::istream& operator>>(std::istream& aIStream, Polynomial& aObject)
{
    aIStream >> aObject.fDegree;

    for (int i = static_cast<int>(aObject.fDegree); i >= 0; i--)
    {
        aIStream >> aObject.fCoeffs[i];
    }

    return aIStream;
}

// Output operator for polynomials (highest to lowest)
std::ostream& operator<<(std::ostream& aOStream, const Polynomial& aObject)
{
    bool first = true;

    for (int i = static_cast<int>(aObject.fDegree); i >= 0; i--)
    {
        double coeff = aObject.fCoeffs[i];

        if (coeff == 0.0)
            continue;

        if (!first && coeff > 0)
            aOStream << "+";

        aOStream << coeff << "x^" << i;
        first = false;
    }

    if (first)
        aOStream << "0";

    return aOStream;
}

// ----- PROBLEM SET 1 EXTENSIONS -----

// Call operator to evaluate polynomial at x
double Polynomial::operator()(double aX) const
{
    double result = 0.0;

    for (int i = static_cast<int>(fDegree); i >= 0; i--)
    {
        result += fCoeffs[i] * pow(aX, i);
    }

    return result;
}

// Compute derivative of the polynomial
Polynomial Polynomial::getDerivative() const
{
    Polynomial result;

    if (fDegree == 0)
    {
        result.fDegree = 0;
        result.fCoeffs[0] = 0.0;
        return result;
    }

    result.fDegree = fDegree - 1;

    for (size_t i = 1; i <= fDegree; i++)
    {
        result.fCoeffs[i - 1] = fCoeffs[i] * static_cast<double>(i);
    }

    return result;
}

// Compute indefinite integral of the polynomial
Polynomial Polynomial::getIndefiniteIntegral() const
{
    Polynomial result;
    result.fDegree = fDegree + 1;

    for (size_t i = 0; i <= fDegree; i++)
    {
        result.fCoeffs[i + 1] = fCoeffs[i] / static_cast<double>(i + 1);
    }

    result.fCoeffs[0] = 0.0; // Constant of integration

    return result;
}

// Compute definite integral from aXLow to aXHigh
double Polynomial::getDefiniteIntegral(double aXLow, double aXHigh) const
{
    Polynomial integral = getIndefiniteIntegral();
    return integral(aXHigh) - integral(aXLow);
}
```