Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: 1, Solution Design in C++
Due date: Sunday, January 25, 2026, 23:59
Lecturer: Dr. Bui Van Vuong

Your name: Nguyen Duc Manh **Your student ID:** 105547489

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Tues 16:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30

Marker's comments:

Problem	Marks	Obtained
1	38	
2	60	
3	38	
4	20	
Total	156	

Extension certification:

This assignment has been given an extension and is now due on _____

Signature of Convener:_____

```cpp
#include "Polygon.h"
#include <cmath>

float Polygon::getSignedArea() const
{
    float area = 0.0f;
    float abs_sum = 0.0f;

    for (int i = 0; i < fNumberOfVertices; i++) {
        Vector2D firstVertex = getVertex(i);
        int nextVertexIndex = (i + 1) % fNumberOfVertices;
        Vector2D secondVertex = getVertex(nextVertexIndex);
        float step = firstVertex.cross(secondVertex);
        abs_sum += step;
    }

    area = abs_sum / 2;
    return area;
}
```

```cpp

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


double Polynomial::operator()(double aX) const
{
    double result = 0.0;
    // Replace each terms with input value of x and calculate
    // Therefore we don't have to reverse the for loop
    for (size_t i = 0; i <= fDegree; i++) {
        result += fCoeffs[i] * pow(aX, i);
    }
    return result;
};

Polynomial Polynomial::getDerivative() const
{

    if (fDegree == 0) {
        return Polynomial();
    }

    Polynomial result;
    result.fDegree = fDegree - 1;
    for (size_t i = 1; i <= fDegree; i++) {
        // The 0 term will disappear
        size_t newTerm = i - 1;
        double newCoeff = fCoeffs[i] * i;
        result.fCoeffs[newTerm] = newCoeff;
    }

    return result;
};

Polynomial Polynomial::getIndefiniteIntegral() const
{
    Polynomial result;

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

    return result;
};

```

```
50 double Polynomial::getDefiniteIntegral(double aXLow, double aXHigh) const
51 {
52     double result = 0.0;
53     Polynomial afterInIntegral = getIndefiniteIntegral();
54     double lowValue = afterInIntegral.operator()(aXLow);
55     double highValue = afterInIntegral.operator()(aXHigh);
56     result = highValue - lowValue;
57     return result;
58 };
```

```cpp
#include "Combination.h"
#include <cstddef>


Combination::Combination(size_t aN, size_t aK) : fN(aN), fK(aK) {};
size_t Combination::getN() const { return fN; };
size_t Combination::getK() const { return fK; };

unsigned long long Combination::operator()() const {
    if (fN < fK) { return 0ULL; }
    unsigned long long result = 1ULL;
    // Now the index is = fK
    for (size_t i = 1; i <= fK; i++) {
        unsigned long long divisor = fN - (i - 1);
        unsigned long long dividend = i;
        result = (result * divisor) / dividend;

    }
    return result;
};
```

```cpp
#include "BernsteinBasisPolynomial.h"
#include <cmath>

// my code

BernsteinBasisPolynomial::BernsteinBasisPolynomial(unsigned int aV,
  unsigned int aN) : fFactor(aN, aV) {};

double BernsteinBasisPolynomial::operator()(double aX) const {
    unsigned int aV = fFactor.getK();
    unsigned int aN = fFactor.getN();

    if ((aX == 0.0 && aV != 0) || (aX == 1 && aV != aN)) {
        return 0.0;
    }

    double result = 0.0;

    double module1 = fFactor();
    double module2 = pow(aX, aV);
    double module3 = pow((1 - aX), (aN - aV));

    result = module1 * module2 * module3;

    return result;
};
```