COS30008 Semester 1, 2024 Dr. James Jackson

# 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:** Friday, February 2, 2024, 23:59

**Lecturer:** Dr. James Jackson

## Your name: Nguyen Gia Binh Your student ID: 104219428

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | Sat 10:00 |
|  |  |  |  |  |  |  |  |  |  | X |

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:

PolygonPS1.cpp:

#include "Polygon.h"

float Polygon::getSignedArea() const {

    float lResult = 0.0;

    for (size\_t i = 0; i < fNumberOfVertices; i++)

    {

        size\_t lj = (i + 1) % fNumberOfVertices;

        lResult += (fVertices[i].getX() \* fVertices[lj].getY() - fVertices[i].getY() \* fVertices[lj].getX());

    }

    lResult \*= 0.5;

    return lResult;

}

PolynomialPS1.cpp:

#include "Polynomial.h"

#include <cmath>

double Polynomial::operator()(double aX) const {

    double lResult = 0.0;

    for (size\_t i = 0; i <= fDegree; i++) {

        lResult += fCoeffs[i] \* pow(aX, i);

    }

    return lResult;

}

Polynomial Polynomial::getDerivative() const {

    Polynomial lDerivative;

    lDerivative.fDegree = fDegree - 1;

    for (size\_t i = 1; i <= fDegree; i++) {

        //if (i != 0) {

            lDerivative.fCoeffs[i - 1] = fCoeffs[i] \* i;

        //}

        //else {

        //  lDerivative.fCoeffs[i - 1] = 0;

        //}

        //in this case the equation return back is correct according to the example in pdf

        //but the output still print error "Derivative is broken".

        //i can be 0 but i dont know why when i put i = 0 it broke the derivative of indefinite integral

    }

    return lDerivative;

}

Polynomial Polynomial::getIndefiniteIntegral() const {

    Polynomial lIndefiniteIntegral;

    lIndefiniteIntegral.fDegree = fDegree + 1;

    for (size\_t i = 0; i <= fDegree; i++) {

        lIndefiniteIntegral.fCoeffs[i + 1] = fCoeffs[i] / (i + 1);

    }

    return lIndefiniteIntegral;

}

double Polynomial::getDefiniteIntegral(double aXLow, double aXHigh) const {

    double lLowerBound;

    double lHigherBound;

    lLowerBound = getIndefiniteIntegral()(aXLow);

    lHigherBound = getIndefiniteIntegral()(aXHigh);

    return lHigherBound - lLowerBound;

}

Combination.cpp:

#include "Combination.h"

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 (fK > fN) {

        return 0;

    }

    unsigned long long lResult = 1;

    for (size\_t i = 0; i < fK; i++) {

        //float would work here too but i can see that result is in long long so i use double to have a more accurate decimal

        //which does not matter in this case but just to be save.

        lResult \*= static\_cast<double>(fN - i) / (i + 1);

    }

    return lResult;

}

BernSternBasisPolynomial.cpp:

#include "BernsteinBasisPolynomial.h"

#include "Combination.h"

#include <cmath>

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

}

//double BernsteinBasisPolynomial::operator()(double aX) const {

//  double lResult;

//  lResult = fFactor() \* pow(aX, fFactor.getK()) \* pow((1 - aX), (fFactor.getN() - fFactor.getK()));

//  return lResult;

//}

//i was going with the one liner but after reading the Preliminaries in the pdf again, the requirement seems to emphasize on the for-loop

//so i adapt the code and it return the same result as the example in the pdf

double BernsteinBasisPolynomial::operator() (double aX) const {

    double lResult = fFactor();

    for (size\_t i = 0; i < fFactor.getK(); i++) {

        lResult \*= pow(aX, 1.0);

    }

    for (size\_t i = 0; i < fFactor.getN() - fFactor.getK(); i++) {

        lResult \*= pow(1 - aX, 1.0);

    }

    return lResult;

}