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: Thursday, March 24, 2022, 14:30

Lecturer: Dr. Markus Lumpe

Your name: Nguyen Duc Chung

Your student ID: 104972970

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
					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:__________________________

Problem 1:

```
//PolygonPS1.cpp
#include "Polygon.h"
#include <cmath>


float Polygon::getSignedArea() const
{
   float area = 0.0f;  // Initialize area to zero

   // Ensure the polygon has at least 3 vertices
   if (fNumberOfVertices > 2)
   {

      for (size_t i = 0; i < fNumberOfVertices - 1; ++i)
      {
         // This part prepare values for terms like x1.y2 and y1.x2
         float nowX = fVertices[i].getX();
         float nowY = fVertices[i].getY();
         float nextX = fVertices[i + 1].getX();
         float nextY = fVertices[i + 1].getY();

         // Return the positive part: x1.y2 + x2.y3 + x3.y1
         area = area + (nowX * nextY);
         // Return the negative part: -( y1.x2 + y2.x3 + y3.x1 )
         area = area - (nextX * nowY);
      }
         //  This handles the terms for xn.y1 and yn.x1 , closing the loop
         float lastX = fVertices[fNumberOfVertices - 1].getX();
         float lastY = fVertices[fNumberOfVertices - 1].getY();
         float firstX = fVertices[0].getX();
         float firstY = fVertices[0].getX();

         area = area + (lastX * firstY);
         area = area - (firstX * lastY);

   }

  // the calculated area is divided by 2 to get correct value.
   return area * 0.5f;
}
```

Problem 2:

```
// PolynomialPS1.cpp
#include "Polynomial.h"
#include <cmath> // For pow
#include <iostream>
using namespace std;

// Default constructor
Polynomial::Polynomial() : fDegree(0) {
    for (size_t i = 0; i <= MAX_DEGREE; ++i) {
        fCoeffs[i] = 0.0;
    }
}

// Operator to multiply two polynomials
Polynomial Polynomial::operator*(const Polynomial& aRHS) const {
    Polynomial result;
    result.fDegree = fDegree + aRHS.fDegree; // Degree of product

    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]; // Multiply terms
        }
    }

    return result;
}

// Operator to compare two polynomials
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)
istream& operator>>(istream& aIStream, Polynomial& aObject) {
    cout << "Enter the degree of the polynomial: ";
    aIStream >> aObject.fDegree;

    if (aObject.fDegree > MAX_POLYNOMIAL) {
        cout << "Degree exceeds maximum allowed, setting degree to " << MAX_POLYNOMIAL <<
endl;
        aObject.fDegree = MAX_POLYNOMIAL;
    }
    for (int i = aObject.fDegree; i >= 0; --i) {
        cout << "Enter coefficient for x^" << i << ": ";
        aIStream >> aObject.fCoeffs[i];
    }

    return aIStream;
}

// Output operator for polynomials
ostream& operator<<(ostream& aOStream, const Polynomial& aObject) {
    for (int i = aObject.fDegree; i > 0; --i) {
        aOStream << aObject.fCoeffs[i] << "x^" << i << " + ";
    }
    aOStream << aObject.fCoeffs[0];
    return aOStream;
}

// Call operator to calculate the polynomial for a given x
double Polynomial::operator()(double aX) const {
    double result = 0.0;
    for (size_t i = 0; i <= fDegree; ++i) {
        result += fCoeffs[i] * pow(aX, i);  // Apply aX raised to the power of i
    }
    return result;
}

// Compute derivative: the derivative is a fresh polynomial with degree fDegree-1
Polynomial Polynomial::getDerivative() const {
    Polynomial derivative;
    if (fDegree == 0) {
        return derivative;  // Derivative of a constant is zero
    }
    derivative.fDegree = fDegree - 1;
    for (size_t i = 1; i <= fDegree; ++i) {
        derivative.fCoeffs[i - 1] = fCoeffs[i] * i;  // Coeff * degree
    }
    return derivative;
}

// Compute indefinite integral: the indefinite integral is a fresh polynomial with degree fDegree+1
Polynomial Polynomial::getIndefiniteIntegral() const {
    Polynomial integral;
    integral.fDegree = fDegree + 1;
    integral.fCoeffs[0] = 0;  // The constant term (C) is usually set to 0
    for (size_t i = 0; i <= fDegree; ++i) {
        integral.fCoeffs[i + 1] = fCoeffs[i] / (i + 1);
    }
    return integral;
}

// Calculate definite integral: computes indefinite integral, evaluates it at xlow and xhigh
double Polynomial::getDefiniteIntegral(double aXLow, double aXHigh) const {
    Polynomial integral = getIndefiniteIntegral();
    return integral(aXHigh) - integral(aXLow);
}
```

Problem 3:

Combination.h

```
#pragma once

// COS30008, Problem Set 1/3, 2022
#pragma once
#include <cstddef>
class Combination
{
private:
size_t fN;
size_t fK;
public:
// constructor for combination n over k with defaults
Combination(size_t aN = 0, size_t aK = 0) : fN(aN), fK(aK) {};
size_t getN() const { return fN; } ;
size_t getK() const { return fK; } ;
// call operator to calculate n over k
// We do not want to evaluate factorials.
// Rather, we use this method
//
// n (n-0) (n-1) (n - (k - 1))
// ( ) = ----- * ----- * ... * -------------
// k 1 2 k
//
// which maps to a simple for-loop over 64-bit values.
unsigned long long operator()() const
{
if (fK > fN) // Return 0 if k is greater than n
return 0;

unsigned long long result = 1;

for (size_t i = 0; i < fK; ++i)
{
result *= (fN - i);    // Multiply by (n - i)
result /= (i + 1);     // Divide by (i + 1)
}

return result;
}
};
```

Problem 4:
BernsteinBasisPolynomial.h:

```
#pragma once

// COS30008, Problem Set 1/4, 2022
#pragma once
#include "Combination.h"
class BernsteinBasisPolynomial
{
private:
                                         Combination fFactor;
public:
                                 // constructor for b(v,n) with defaults
    BernsteinBasisPolynomial(unsigned int aV = 0, unsigned int aN = 0) : fFactor(aN, aV) {};
                               // call operator to calculate Berstein base
                                  // polynomial for a given x (i.e., aX)
                                  double operator()(double aX) const
   {
      if (aX < 0.0 || aX > 1.0) // Ensure x is within [0, 1]
         return 0.0;

      unsigned int v = fFactor.getK();
      unsigned int n = fFactor.getN();

      double term1 = static_cast<double>(fFactor());
      double term2 = pow(aX, v);
      double term3 = pow(1.0 - aX, n - v);
      return term1 * term2 * term3;
   };
};
```