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, October 08, 2023, 23:59 (VN Time)

Lecturer: Dr. Van Dai PHAM

Your name: Nguyen Quoc Thang **Your student ID:** 104193360

Check Tutorial	Mon 10:00	Tues 10:30	Tues 12:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30	Thurs 10:00 Innovation Lab	Thurs 14:00	Frid 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<iostream>
#include"Polygon.h" ;

using namespace std;

float Polygon::getSignedArea() const {
  float signArea = 0.0f;
  float result = 0.0f;
  for (int i = 0; i < fNumberOfVertices; i++) {
    int nexttop = (i + 1) % fNumberOfVertices;
    signArea += ((fVertices[i].getX() * fVertices[nexttop].getY()) - (fVertices[i].getY() * fVertices[nexttop].getX()));
    result = signArea / 2 ;
  }
  return result;
}
```

PolynomialPS1.cpp

```
#include<iostream>
#include<cmath>
#include"Polynomial.h"
using namespace std;
double Polynomial::operator()(double aX) const {

  double result = 0.0;

  for (int i = 0; i <= fDegree; i++)
  {
    result += fCoeffs[i] * pow(aX, i);
  }
  return result;
}
Polynomial Polynomial::getDerivative() const
{
  Polynomial derivative;

  derivative.fDegree = fDegree - 1;
  for (int i = 0; i <= fDegree; i++)
  {
    if (i == 0) {
        derivative.fCoeffs[i] = 0 ;
    }else
    {
      derivative.fCoeffs[i - 1] = fCoeffs[i] * i ;
    }
  }

  return derivative;
}
Polynomial Polynomial::getIndefiniteIntegral() const {
  Polynomial indef;
  indef.fDegree = fDegree + 1;
  for (int i = 0; i <= indef.fDegree; i++) {
    indef.fCoeffs[i + 1] = (fCoeffs[i] / (i + 1));
  }
  return indef;
}
double Polynomial::getDefiniteIntegral(double aXLow, double aXHigh) const {
  double lowresult = 0.0;
  double highresult = 0.0;
  //define.fDegree = fDegree + 1;
  Polynomial define = getIndefiniteIntegral();
  for (int i = 0; i <= define.fDegree; i++) {
    //define.fCoeffs[i + 1] = (fCoeffs[i] / (i + 1));
    lowresult += define.fCoeffs[i + 1] * pow(aXLow, (i + 1));
    highresult += define.fCoeffs[i + 1] * pow(aXHigh, (i + 1));
  }
  double result = highresult - lowresult;
  return result;
}
```

Combination.cpp,

```
#include<iostream>
#include"Combination.h"
using namespace std;
using ll = long long ;
Combination::Combination(size_t aN , size_t aK )  {
  this->fN = aN;
  this->fK = aK;
}
size_t Combination::getN() const {
  return this->fN ;
}
size_t Combination::getK() const {
  return this->fK;
}
unsigned long long Combination::operator()() const {
  unsigned long long result = 1;
  float numerator = 0.0f;
  if (fK > fN) {
    return 0 ;
  }

  for (size_t i = 1; i <= fK; i++) {
    numerator =  fN - (i - 1) ;
    result *= (numerator / i);
    }
  return result;
}
```

BersteinBasisPolynomial.cpp

```
#include<iostream>
#include"BernsteinBasisPolynomial.h"
#include<cmath>
using namespace std;

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

}
double BernsteinBasisPolynomial::operator()(double aX) const {
  /*unsigned long long  combination = 1;
  double numerator = 0.0f;
  for (size_t i = 1; i <= fFactor.getK(); i++) {
    numerator = fFactor.getN() - (i - 1);
    combination *= (numerator / i);
  }*/
  double result = 0.0f;
  result = fFactor() * pow(aX, fFactor.getK()) * pow((1 - aX), (fFactor.getN() - fFactor.getK()));
  return result;
}
```