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:	March 30, 2021, 16:00
Lecturer:	Dr. Markus Lumpe

Your name:____________________________ **Your student ID:**_______________

Check Tutorial	Wed 08:30	Wed 10:30	Wed 16:30	Thurs 08:30	Thurs 10:30	Thurs 14:30	Thurs 16:30	Fri 08:30	Fri 10:30	Fri 14:30

Marker's comments:

Problem	Marks	Obtained
1	16+19+17+8 = 60	
2	38	
3	22	
Total	120	

Extension certification:

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

Signature of Convener:_________________________

Problem Set 1: Solution Design in C++

Preliminaries

The goal of this problem set is to extend the solution of tutorial 3. In particular, we wish to add methods to calculate a polynomial and to determine a polynomial's derivative and it's indefinite and definite integral. In addition, we look at Bernstein polynomials that form the basis for Bézier curves, parametric curves used in computer graphics and engineering design.

Remember, a polynomial can be expressed concisely by using the summation notation:

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

That is, a polynomial can be written as the sum of a finite number of terms $a_i x^i$. Each term consists of the product of a number a_i, called the coefficient, and a variable x raised to integer powers – x^i. The exponent i in x^i is called the degree of the term $a_i x^i$. The degree of a polynomial is the largest exponent of any one term with a non-zero coefficient.

To calculate a polynomial, we need to provide a value for the variable x. Again, this allows for a straightforward mapping to a for-loop is C++. To compute the x^i, we need to use the function `pow`, called *raise-to-power*, which is defined in `cmath`. For more details and uses of `pow`, see http://www.cplusplus.com/reference/cmath/pow/ or Microsoft's documentation at https://docs.microsoft.com/en-us/cpp/standard-library/cmath?view=msvc-160.

Please refer to PolynomialMath.pdf available on Canvas to review how one can compute the differential and integral of a polynomial. An understanding of the mathematical principles is essential for the successful completion of this assignment.

Conditional Compilation

The test driver provided for this problem set makes use of conditional compilation via preprocessor directives. This allows you to focus only on the task you are working on.

This problem set is comprised of three problems. The test driver (i.e., main.cpp) uses `P1`, `P2`, and `P3` as variables to enable/disable the test associated with a corresponding problem. To enable a test just uncomment the respective `#define` line. For example, to test problem 2 only, enable #define P2:

```
// #define P1
#define P2
// #define P3
```

In Visual Studio, the code blocks enclosed in **`#ifdef`** `PX` … **`#endif`** are grayed out, if the corresponding test is disabled. The preprocessor definition **`#ifdef`** `PX` … **`#endif`** enables conditional compilation. XCode does not use this color coding scheme.

In addition, the test driver defines **`#define`** `Automate` and **`#ifdef`** `Automate` … **`#endif`**. If you enable `Automate`, then the test driver will simulate the input for problem 1.

Problem 1

Start with the solution of tutorial 3 (see Canvas). Do not edit the provided files. Rather create a new .cpp unit, called `PolynomialPS1.cpp`, to implement the new methods. Please note that in C++ the implementation of a class does not need to occur in just one compilation unit (i.e., a .cpp file). As long as the compiler and the linker see all definitions, the build process should succeed, if the program does not contain any errors.

The extended specification of class Polynomial is shown below. You only need to implement the last four methods. The other features (i.e., constructor and operators) are given as part of the solution for tutorial 3. In the .cpp file for the new methods you need to include `cmath` that contains the definition of `pow` – raise to power.

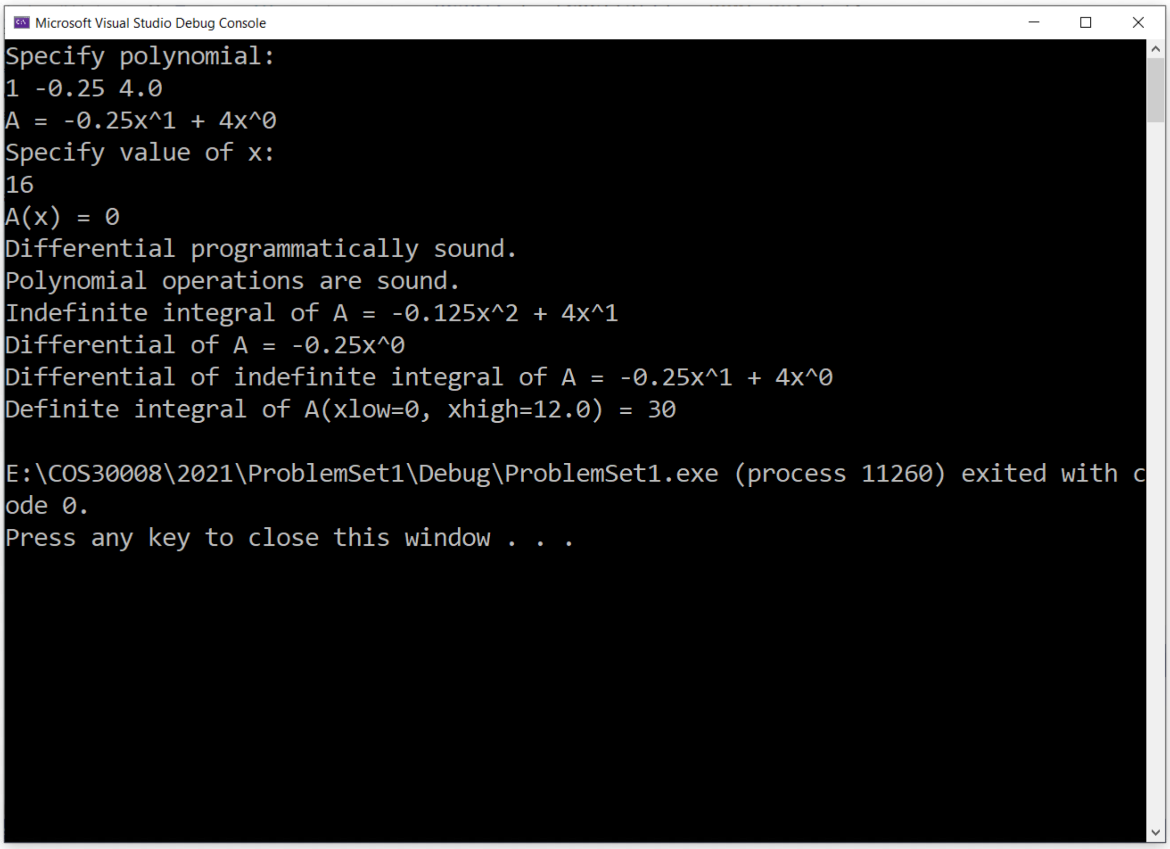
```
#pragma once

#include <iostream>

#define MAX_POLYNOMIAL 10               // max degree for input
#define MAX_DEGREE MAX_POLYNOMIAL*2+1   // max degree = 10 + 10 + 1 = 21

class Polynomial
{
private:
  int fDegree;                        // the maximum degree of the polynomial
  double fCoeffs[MAX_DEGREE+1];       // the coefficients (0..10, 0..20)

public:

  // the default constructor (initializes all member variables)
  Polynomial();

  // binary operator* to multiple to polynomials
  // arguments are read-only, signified by const
  // the operator* returns a fresh polynomial with degree i+j
  Polynomial operator*( const Polynomial& aRight ) const;

  // binary operator== to compare two polynomials
  // arguments are read-only, signified by const
  // the operator== returns true if this polynomial is
  // structurally equivalent to the aRHS polynomial
  bool operator==( const Polynomial& aRHS ) const;

  // input operator for polynomials
  friend std::istream& operator>>( std::istream& aIStream,
                                   Polynomial& aObject );

  // output operator for polynomials
  friend std::ostream& operator<<( std::ostream& aOStream,
                                   const Polynomial& aObject );

  // new methods in problem set 1

  // call operator to calculate polynomial for a given x (i.e., aX)
  double operator()( double aX ) const;

  // compute differential: the differential is a fresh polynomial with degree
  // fDegree-1, the method does not change the current object
  Polynomial getDifferential() const;
  // compute indefinite integral: the indefinite integral is a fresh
  // polynomial with degree fDegree+1
  // the method does not change the current object
  Polynomial getIndefiniteIntegral() const;

  // calculate definite integral: the method does not change the current
  // object; the method computes the indefinite integral and then
  // calculates it for xlow and xhigh and returns the difference
  double getDefiniteIntegral( double aXLow, double aXHigh ) const;
};
```

Please note the changed value of `MAX_DEGREE`. Since we wish to build integrals, we need to reserve one more slot it the array for the coefficients of a polynomial. Our program supports polynomials up to the 10th degree. Multiplying them results in a polynomial up to the 20th degree, whose integral is up to the 21st degree. Hence, we need 22 entries in the array of coefficients (21+1).

Use as main program the following code:

```
#include <iostream>

#include "Polynomial.h"

using namespace std;

void runProblem1()
{
  Polynomial A;
  cout << "Specify polynomial:" << endl;
  cin >> A;
  cout << "A = " << A << endl;


  double x;
  cout << "Specify value of x:" << endl;
  cin >> x;

  cout << "A(x) = " << A(x) << endl;

  Polynomial B;

  if (B == B.getDifferential())
  {
    cout << "Differential programmatically sound." << endl;
  }
  else
  {
    cout << "Differential is broken." << endl;
  }

  if (A == A.getIndefiniteIntegral().getDifferential())
  {
    cout << "Polynomial operations are sound." << endl;
  }
  else
  {
    cout << "Polynomial operations are broken." << endl;
  }
  cout << "Indefinite integral of A = "
       << A.getIndefiniteIntegral() << endl;

  cout << "Differential of A = "
       << A.getDifferential() << endl;

  cout << "Differential of indefinite integral of A = "
       << A.getIndefiniteIntegral().getDifferential() << endl;

  cout << "Definite integral of A(xlow=0, xhigh=12.0) = "
       << A.getDefiniteIntegral(0, 12.0) << endl;

}
```

Using `-0.25x + 4.0` and `16` for the first calculation, the test code should produce the following result.



```
Specify polynomial:
1 -0.25 4.0
A = -0.25x^1 + 4x^0
Specify value of x:
16
A(x) = 0
Differential programmatically sound.
Polynomial operations are sound.
Indefinite integral of A = -0.125x^2 + 4x^1
Differential of A = -0.25x^0
Differential of indefinite integral of A = -0.25x^1 + 4x^0
Definite integral of A(xlow=0, xhigh=12.0) = 30

E:\COS30008\2021\ProblemSet1\Debug\ProblemSet1.exe (process 11260) exited with c
ode 0.
Press any key to close this window . . .
```

Problem 2

In this task, we define a simple data type to represent combinations

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

A combination is a selection of items from a collection, such that the order of selection does not matter.

Here, we are interested only in the binomial coefficient

$$\frac{n!}{k!(n-k)!}$$

Whenever k ≤ n, and which is zero when k > n.

Define a C++ class that implements combination. The specification of class `Combination` is shown below:

```
#pragma once

class Combination
{
private:
  unsigned int fN;
  unsigned int fK;

public:

  // constructor for combination n over k
  Combination( unsigned int aN, unsigned int aK );

  // getters
  unsigned int getN() const;
  unsigned int getK() const;

  // 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.
  // https://en.wikipedia.org/wiki/Combination
  unsigned long long operator()() const;
};
```

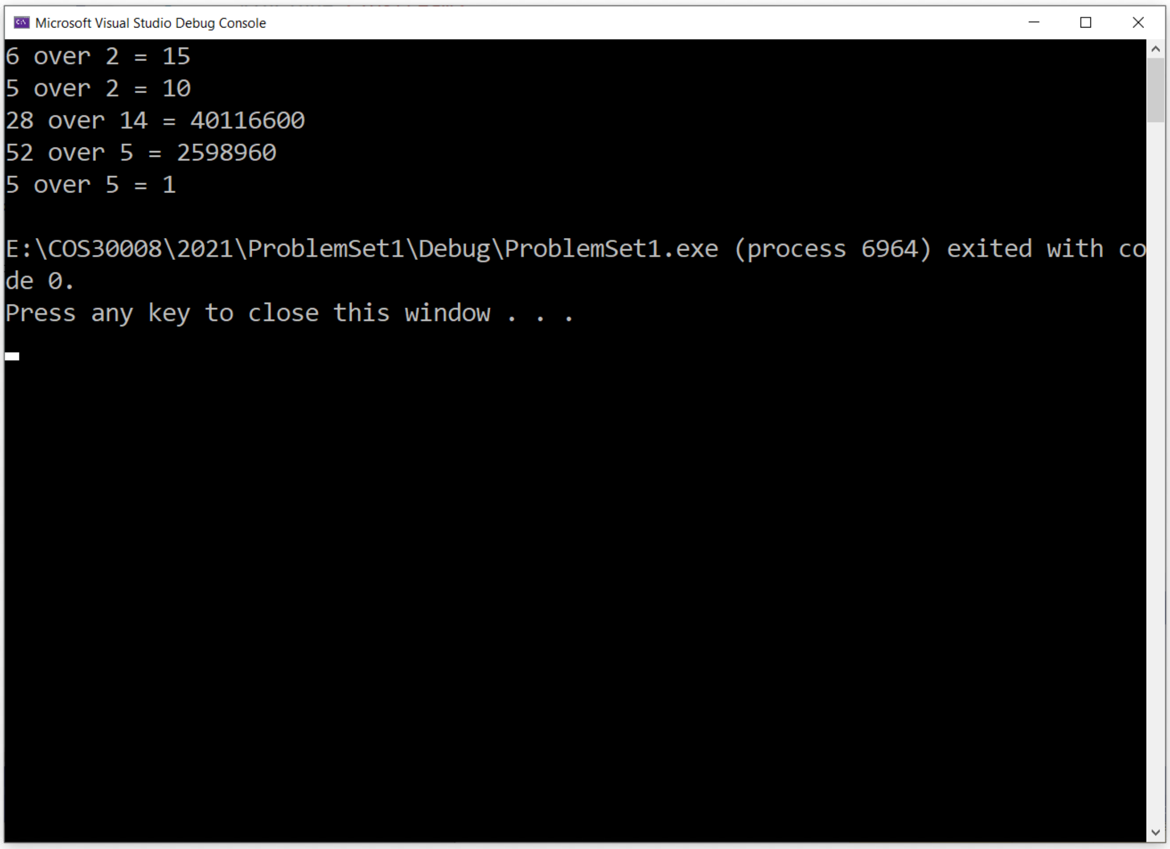
The class Combination has two instance variables representing n and k. The constructor initializes those instance variables and the getters provide read-only access to their values.

The call `operator()` returns the value of a combination. We use the method described on WikiPedia: https://en.wikipedia.org/wiki/Combination rather than calculating the factorials, which can become very big numbers. You need to use the type **`unsigned long long`** for calculation in order to work with 64-bit values.

To test your implementation of class `Combination`, we can use the following test driver.

```
#include <iostream>

#include "Combination.h"

void runProblem2()
{
  Combination a(6, 2);
  Combination b(5, 2);
  Combination c(28, 14);
  Combination d(52, 5);
  Combination e(5, 5);

  cout << a.getN() << " over " << a.getK() << " = " << a() << endl;
  cout << b.getN() << " over " << b.getK() << " = " << b() << endl;
  cout << c.getN() << " over " << c.getK() << " = " << c() << endl;
  cout << d.getN() << " over " << d.getK() << " = " << d() << endl;
  cout << e.getN() << " over " << e.getK() << " = " << e() << endl;
}
```

The test code should produce the following result.



```
6 over 2 = 15
5 over 2 = 10
28 over 14 = 40116600
52 over 5 = 2598960
5 over 5 = 1

E:\COS30008\2021\ProblemSet1\Debug\ProblemSet1.exe (process 6964) exited with co
de 0.
Press any key to close this window . . .
```

Problem 3

Bézier curves are a fundamental tool in computer graphics and engineering design. They are named after Pierre Bézier who developed them for designing curves for the bodywork of Renault cars. A central building block of Bézier curves is a special type of polynomial – Bernstein polynomials (see https://en.wikipedia.org/wiki/Bernstein_polynomial) – that is restricted to the interval [0,1]. Bernstein polynomials serve as a blending function between control points that are used to interpolate a curve.

In this problem, we only consider Bernstein base polynomials, which are defined as follows:

$$b_{v,n}(x) = \binom{n}{v} x^v (1-x)^{n-v}, \qquad v = 0, \ldots, n$$

where $\binom{n}{v}$ is a binomial coefficient. For example

$$b_{2,5}(x) = \binom{5}{2} x^2 (1-x)^{5-2} = 10x^2(1-x)^3$$

Every Bernstein base polynomial $b_{v,n}(x)$ creates a curve in the interval [0,1].

Define a C++ class that implements Bernstein base polynomials. The specification of class `BernsteinBasePolynomial` is shown below

```
#pragma once

#include "Combination.h"

// https://en.wikipedia.org/wiki/Bernstein_polynomial
class BernsteinBasePolynomial
{
private:
  Combination fFactor;

public:

  // constructor for b(0,0)
  BernsteinBasePolynomial();

  // constructor for b(v,n)
  BernsteinBasePolynomial( unsigned int aV, unsigned int aN );

  // call operator to calculate Berstein base
  // polynomial for a given x (i.e., aX)
  double operator()( double aX ) const;
};
```

The class `BernsteinBasePolynomial` has one instance variable to represent the required binomial coefficient. There are two overloaded constructors: a default constructor and a constructor with two arguments. Please overserve the order which is different compared to that in class `Combination`.

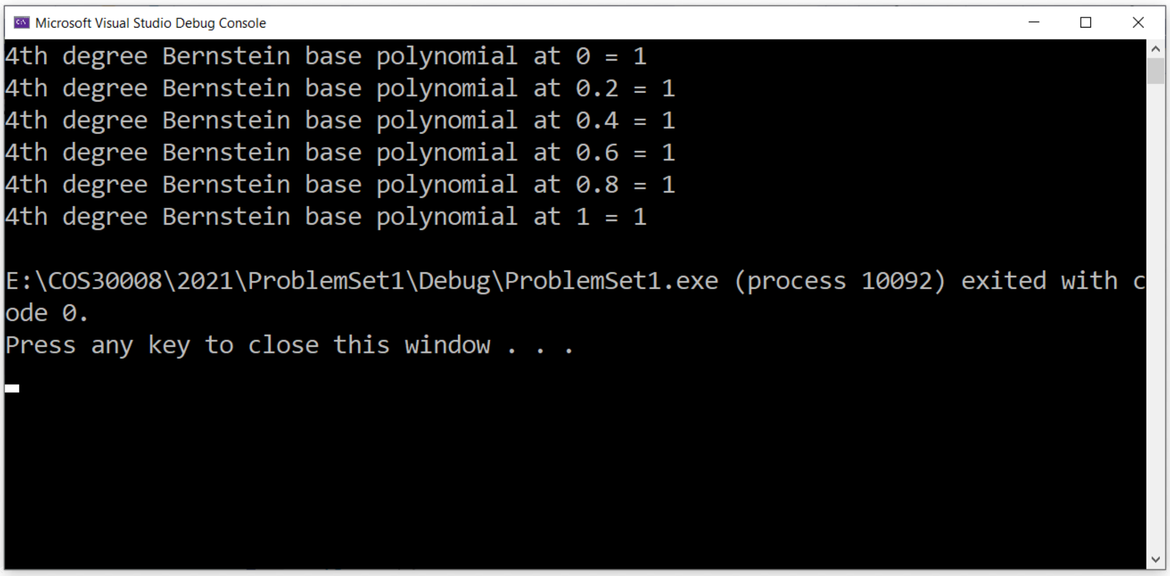
The call **`operator()`** evaluates Bernstein base polynomial object for a given value of `aX`. We assume that `aX` is always in the interval [0,1].

To test your implementation of class `BernsteinBasePolynomial`, we can use the following test driver.

```
#include <iostream>

#include "BernsteinBasePolynomial.h"

void runProblem3()
{
  BernsteinBasePolynomial bba(0, 4);
  BernsteinBasePolynomial bbb(1, 4);
  BernsteinBasePolynomial bbc(2, 4);
  BernsteinBasePolynomial bbd(3, 4);
  BernsteinBasePolynomial bbe(4, 4);

  for (double i = 0.0; i < 1.1; i += 0.2)
  {
    cout << "4th degree Bernstein base polynomial at "
         << i << " = "
         << bba(i) + bbb(i) + bbc(i) + bbd(i) + bbe(i)
         << endl;
    }
}
```

The test code should produce the following result. Please note that the test code interpolates a line made of curve points between x in [0,1] that gives y = 1:

$$\sum_{v=0}^{4} b_{v,4}(x) = 1, \qquad \forall x.\ 0 \leq x \leq 1$$



```
4th degree Bernstein base polynomial at 0 = 1
4th degree Bernstein base polynomial at 0.2 = 1
4th degree Bernstein base polynomial at 0.4 = 1
4th degree Bernstein base polynomial at 0.6 = 1
4th degree Bernstein base polynomial at 0.8 = 1
4th degree Bernstein base polynomial at 1 = 1

E:\COS30008\2021\ProblemSet1\Debug\ProblemSet1.exe (process 10092) exited with c
ode 0.
Press any key to close this window . . .
```

The solution of this problem set requires approx. 120-140 lines of low density C++ code.

Submission deadline: Tuesday, March 30, 2021, 16:00.

Submission procedure: PDF of printed code for code of **`PolynomialPS1`**, **`Combination`**, and **`BernsteinBasePolynomial`**.