Swinburne University of Technology

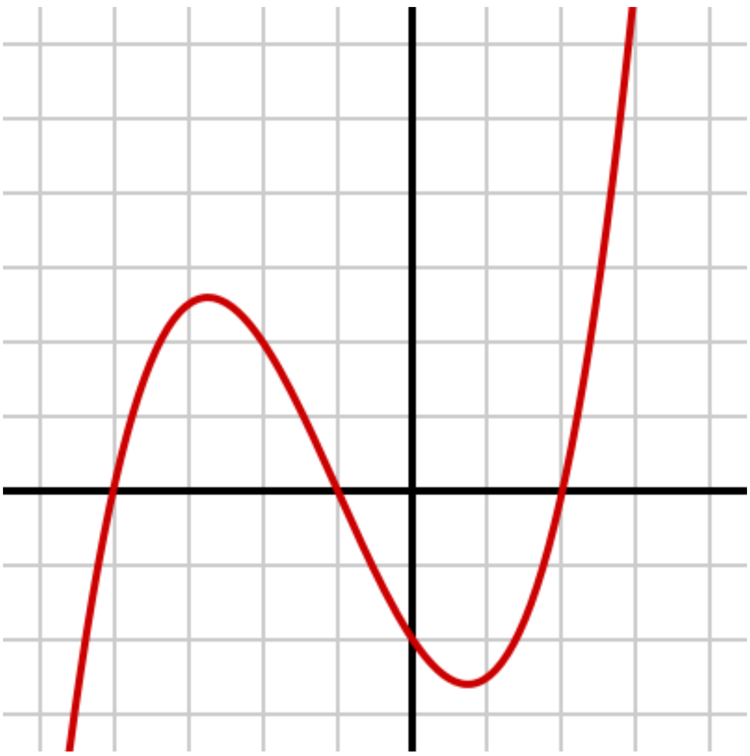
Faculty of Science, Engineering and Technology

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	3, Solution Design in C++
Lecturer:	Dr. Markus Lumpe

However difficult life may seem, there is always something you can do and succeed at.

Steven Hawking



Solution Design in C++

The goal of this laboratory session is to build a C++ console application, called `Polynomials`, that allows users to specify the degree and coefficients of simple polynomials, multiply two polynomials, and output a human-readable representation.

A polynomial with a single variable *x* can be written in the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0 x^0$$

where $0, \ldots, n$ are integers, $a_0, \ldots, a_{\mathrm{n}}$ are real numbers, and x is the variable of the polynomial. Programmatically in C++, we can represent polynomial as an array of double values with length n+1. The degree of a polynomial is the largest exponent of any one term with a non-zero coefficient. For the purpose of this tutorial, we limit the maximum degree of user-specified polynomials to 10.

A polynomial can be expressed more concisely by using summation notation, which allows for a straightforward mapping to a standard for-loop in C++:

$$f(x) = \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0 x^0$$

In C++, we map this summation to a for-loop:

```
for ( int i = 0; i <= n; i++ ) { … }
```

In addition to representing polynomials, we also wish to support polynomial multiplication. Given two polynomials

$$f(x) = \sum_{i=0}^{n} a_i x^i \quad \text{and} \quad g(x) = \sum_{j=0}^{m} b_j x^j$$

the product is defined as

$$f(x)g(x) = \sum_{i=0}^{n} a_i x^i \sum_{j=0}^{m} b_j x^j = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j x^{i+j}$$

In order words, the product of two polynomials can be realized as a nested for-loop that aggregates the respective i^{th} and j^{th} polynomial terms. The maximum degree of the resulting polynomial is i+j. Since we allow 10 as the maximum user-specified degree for polynomials, our implementation must support polynomials up to degree 20 = 10 + 10.

To facilitate the implementation, we shall use fixed-size arrays of double values to represent polynomials. All elements in the array have to be initialized to 0.0. For all non-zero coefficients a_i the array contains at index i the value a_i. As a result, the array arranges a given polynomial from right to left, that is, in increasing degree order.

The application should consist of two parts: a class `Polynomial` that implements the desired functionality plus the equivalence operator == and a `main` function that declares, reads, multiplies polynomials, and outputs the results to the Console. The specification of class `Polynomial` is shown below:

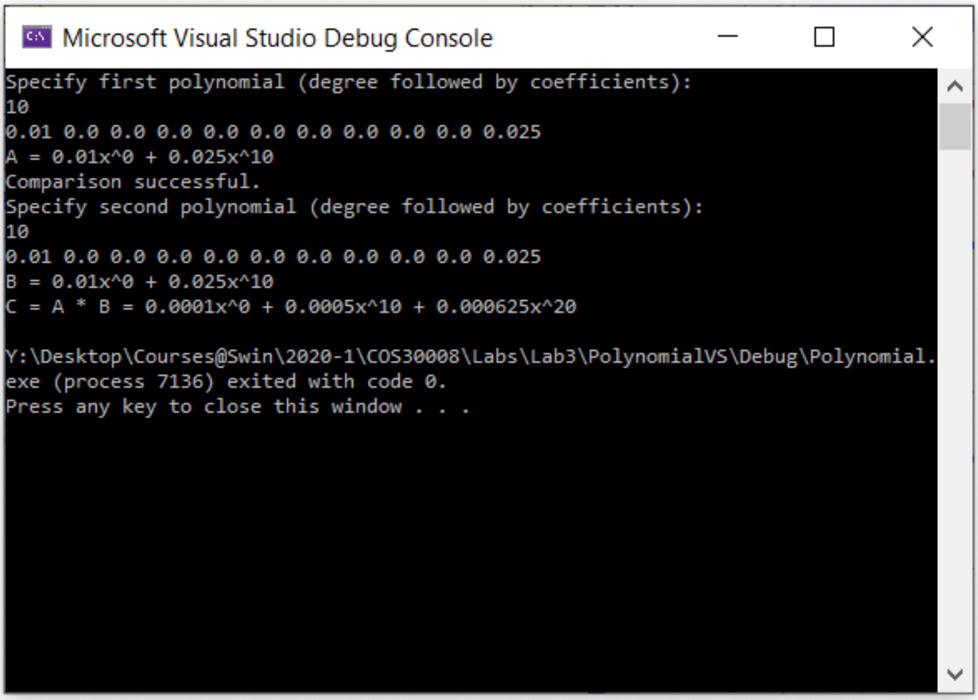
```
#pragma once

#include <iostream>

#define MAX_DEGREE 20+1       // max degree = 10 + 10 + 1, 0 to 20

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

public:

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

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

To implement the class `Polynomial` follow the process outlined in the lecture notes. First implement the constructor. Then implement `operator>>` and `operator<<`. The input operator requires two types of information: the degree (an integer value) and the corresponding number (degree+1) of coefficients (floating-point values). The output operator should only print the polynomial terms with non-zero coefficients. Finally, define the equivalence and multiplication operators.

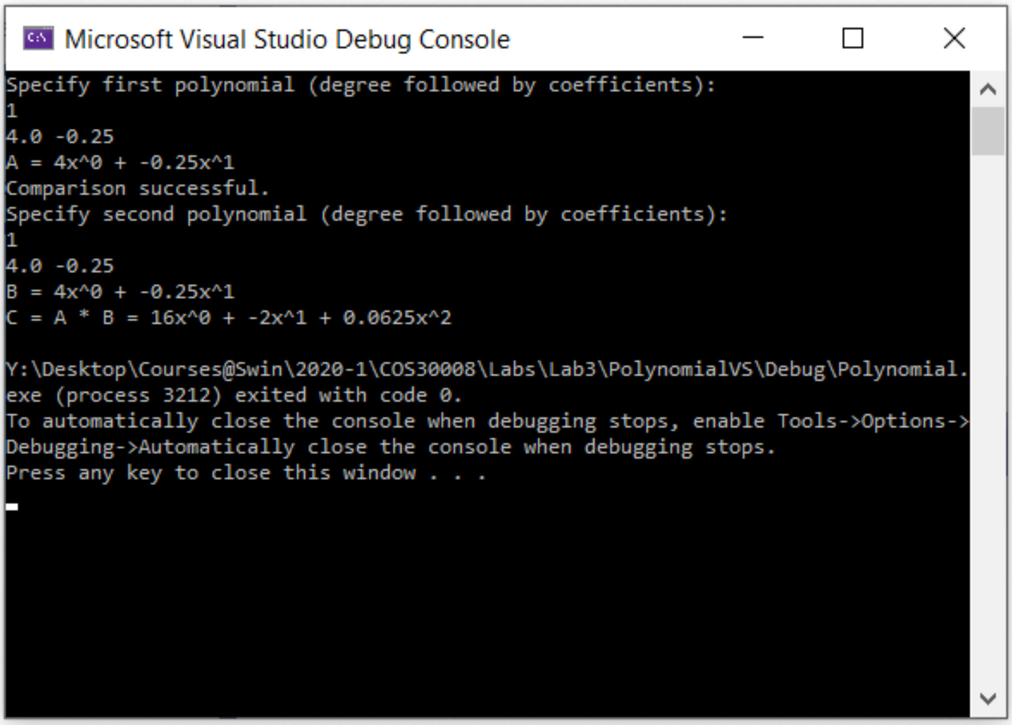
You can use as main program the following code:

```
#include <iostream>

#include "Polynomial.h"

using namespace std;

int main()
{
  Polynomial A;
  cout << "Specify first polynomial (degree followed by coefficients):" << endl;
  cin >> A;
  cout << "A = " << A << endl;

  Polynomial AA = A;

  if ( AA == A )
  {
    cout << "Comparison successful." << endl;
  }
  else
  {
    cout << "Comparison failed." << endl;
  }

  Polynomial B;
  cout << "Specify second polynomial (degree followed by coefficients):" << endl;
  cin >> B;
  cout << "B = " << B << endl;

  Polynomial C = A * B;
  cout << "C = A * B = " << C << endl;

  return 0;
}
```

Naturally, you can comment-out parts that you have not yet implemented. Once the implementation is complete, test your code as shown below (e.g., `-0.25x + 4.0`):



Your solution must support polynomials up to the 10th degree. For example, the polynomial $0.025x^{10} + 0.01$ must produce a result as show below:



```
Microsoft Visual Studio Debug Console
Specify first polynomial (degree followed by coefficients):
1
4.0 -0.25
A = 4x^0 + -0.25x^1
Comparison successful.
Specify second polynomial (degree followed by coefficients):
1
4.0 -0.25
B = 4x^0 + -0.25x^1
C = A * B = 16x^0 + -2x^1 + 0.0625x^2

Y:\Desktop\Courses@Swin\2020-1\COS30008\Labs\Lab3\PolynomialVS\Debug\Polynomial.
exe (process 3212) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

You need to input:

```
10
0.01 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.025
```

The result of the multiplication is a polynomial of the 20th degree: $0.000625x^{20} + 0.0001$.

The solution requires 100-150 lines of low density C++ code.