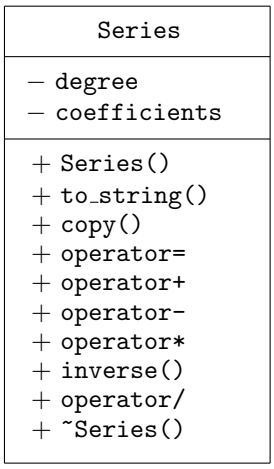
MCS 360 Project Two : computing with truncated power series
due Monday 9 October at 3PM

The main goal of this project is to define and implement a C++ class to compute with truncated power series. The UML class diagram of the `Series` class is drawn below.



The class Series represents a truncated power series with doubles as coefficients. A series $s(t)$ truncated at degree d has $d+1$ coefficients and is written as $s_0+s_1t+s_2t^2+\cdots+s_dt^d+O(t^{d+1})$, where the coefficients s_k are doubles, for k going from 0 to d. This d is called the degree of the series.

The degree and coefficients of a series are private data attributes. The degree is of type `size_t`. The type of the coefficients is `vector<double>`, using the `vector` of the standard template library.

The class has three constructors. The first constructor takes only the degree as input parameter and returns a series with zero coefficients. The second constructor takes as input the degree and the constant leading coefficient of the series. The third constructor also has as input the degree and allows to specify all coefficients of the series. The second parameter is the address of a sequence of doubles, as many as the degree plus one (that is: $d+1$) of the series.

The copy method copies the degree and the coefficients to the object the copy method is applied to. This copy method is used in the definition of the assignment operator.

The addition and subtraction operators are straightforward, defined via the componentwise addition and subtraction. The multiplication of two series is the same as the multiplication of two polynomials, except that coefficients higher than the degree of the series need not be computed.

The multiplicative inverse of a series is defined whenever the constant leading coefficient of the series is nonzero. The inverse $x(t)$ of $s(t)$ is defined via $x(t)*s(t)=1+O(t^{d+1})$. If the series $s(t)$ has coefficients s_k, then the coefficients x_k of the inverse $x(t)$ are computed as

$$\begin{aligned} x_0 &= 1/s_0, \\ x_1 &= -(s_1x_0)/s_0, \\ x_2 &= -(s_1x_1+s_2x_0)/s_0, \\ &\vdots \\ x_d &= -(s_1x_{d-1}+s_2x_{d-2}+\cdots+s_dx_0)/s_0. \end{aligned}$$

The division operator can then be implemented as the product of the numerator by the inverse of the denominator.

The destructor in the class, defined by the `~Series` method, sets the degree of the series to -1.

The arithmetical operations in the class `Series` are tested on random series of a fixed degree. In your implementation of the arithmetical operations you may therefore assume that both operands of the operators are series of the same degree. If the test program is saved in the current working directory under the name `test_series`, then a session at the command prompt `$` in a terminal window could run as follows:

```
$ ./test_series
Testing the arithmetic on truncated power series ...
Give the degree : 3
A random series of degree 3, called A,
 series A : 8.4019e-01-7.8310e-01*t^1-9.1165e-01*t^2+3.3522e-01*t^3 + O(t^4)
and another random series of degree 3, called B,
 series B : -2.7777e-01-4.7740e-01*t^1-3.6478e-01*t^2+9.5223e-01*t^3 + O(t^4)
    A + B : 5.6241e-01-1.2605e+00*t^1-1.2764e+00*t^2+1.2875e+00*t^3 + O(t^4)
A + B - B : 8.4019e-01-7.8310e-01*t^1-9.1165e-01*t^2+3.3522e-01*t^3 + O(t^4)
      A*B : -2.3338e-01-1.8358e-01*t^1+3.2059e-01*t^2+1.4278e+00*t^3 + O(t^4)
    A*B/B : 8.4019e-01-7.8310e-01*t^1-9.1165e-01*t^2+3.3522e-01*t^3 + O(t^4)
```

After the user is prompted for a degree, series with coefficients uniformly distributed in the interval $[-1, +1]$ are generated. Each time the program runs, the user should see different coefficients. The correctness of the operations follows from inspection: we see that `A + B - B` equals `A` and that `A*B/B` equals `A`.

The test program then continues with a test on the square root computation on a random series of degree four. To compute the square root of a series $s(t)$, five steps of Newton's method are applied:

$$f = s(t) - x * x, \quad \Delta x = f/(2 * x), \quad x_{k+1} = x_k + \Delta x, \quad k = 0, 1, \ldots, 4,$$

where $x_0 = s(t)$, i.e.: the sequence starts at $s(t)$.

The program from above then continues in an example session as below:

```
Testing Newton's method on the square root of a random series :
         x = 8.4019e-01-7.8310e-01*t^1-9.1165e-01*t^2+3.3522e-01*t^3-2.7777e-01*t^4 + O(t^5)
at step 0 :
fx = 1.3427e-01+5.3280e-01*t^1+7.0181e-03*t^2-1.6559e+00*t^3-1.1708e-01*t^4 + O(t^5)
dx = 7.9906e-02+3.9155e-01*t^1+4.5582e-01*t^2-1.6761e-01*t^3+1.3889e-01*t^4 + O(t^5)
 x = 9.2009e-01-3.9155e-01*t^1-4.5582e-01*t^2+1.6761e-01*t^3-1.3889e-01*t^4 + O(t^5)
at step 1 :
fx = -6.3850e-03-6.2574e-02*t^1-2.2616e-01*t^2-3.3017e-01*t^3-9.8715e-02*t^4 + O(t^5)
dx = -3.4698e-03-3.5481e-02*t^1-1.3972e-01*t^2-2.5582e-01*t^3-2.2579e-01*t^4 + O(t^5)
 x = 9.1662e-01-4.2703e-01*t^1-5.9554e-01*t^2-8.8213e-02*t^3-3.6468e-01*t^4 + O(t^5)
at step 2 :
fx = -1.2039e-05-2.4622e-04*t^1-2.2285e-03*t^2-1.1690e-02*t^3-3.9241e-02*t^4 + O(t^5)
dx = -6.5671e-06-1.3737e-04*t^1-1.2838e-03*t^2-7.0646e-03*t^3-2.5547e-02*t^4 + O(t^5)
 x = 9.1662e-01-4.2717e-01*t^1-5.9682e-01*t^2-9.5277e-02*t^3-3.9022e-01*t^4 + O(t^5)
at step 3 :
fx = -4.3127e-11-1.8042e-09*t^1-3.5732e-08*t^2-4.4551e-07*t^3-3.9247e-06*t^4 + O(t^5)
dx = -2.3525e-11-9.9514e-10*t^1-1.9970e-08*t^2-2.5297e-07*t^3-2.2719e-06*t^4 + O(t^5)
 x = 9.1662e-01-4.2717e-01*t^1-5.9682e-01*t^2-9.5277e-02*t^3-3.9022e-01*t^4 + O(t^5)
at step 4 :
fx = 0.0000e+00+0.0000e+00*t^1+0.0000e+00*t^2+0.0000e+00*t^3-9.4369e-16*t^4 + O(t^5)
dx = 0.0000e+00+0.0000e+00*t^1+0.0000e+00*t^2+0.0000e+00*t^3-5.1477e-16*t^4 + O(t^5)
 x = 9.1662e-01-4.2717e-01*t^1-5.9682e-01*t^2-9.5277e-02*t^3-3.9022e-01*t^4 + O(t^5)
   sqrt(x) = 9.1662e-01-4.2717e-01*t^1-5.9682e-01*t^2-9.5277e-02*t^3-3.9022e-01*t^4 + O(t^5)
sqrt(x)**2 = 8.4019e-01-7.8310e-01*t^1-9.1165e-01*t^2+3.3522e-01*t^3-2.7777e-01*t^4 + O(t^5)
```

Some important points:

1. You may (not must) collaborate in pairs.
   Both authors will receive the same number of points. A pair consists of two, not three or more.
   The formation of pairs must be done as soon as possible, before September 29, at 3pm.
2. The dialogue in the interactive session should run using the exact same words as shown in the example session on the previous page.
3. Your program will be tested with the `g++` compiler. Points will be deducted if you program does not compile with `g++` or if your `g++` compiled program behaves in an unexpected manner.
4. The solution consists of three files: the header file `series.h`, the implementation `series.cpp`, and a test program `test_series.cpp`. You must define a class. Correct programs without the proper use of a class will receive only half of the points.
5. Every function in your program must have a well documented prototype, placed before the `main()`.
6. Handing in an incomplete but working program is better than handing in a program that crashes or does not run at all.
7. The first line of your C++ program must be

   ```
   // MCS 360 Project Two by <Authors>
   ```

   where you replace the `<Authors>` by your names in alphabetical order.
8. If you work in a pair, then only one author should submit.
9. Email your solution to the project to `janv@uic.edu` before 3PM on Monday 9 October so the date of the email is proof of an on time submission. Bring a printed version of your solution to class.
10. The subject line of your email should be MCS 360 Project Two by `<Authors>` where you replace the `<Authors>` by your names.
11. There is an automatic extension of the deadline till 5PM on the same day. However, late submissions are penalized with ten points off. Submissions after 5PM will not be graded.

If you have questions or difficulties with the project, feel free to come to my office for help.