Linköping Institute of Technology
Department of Science and Technology/ITN
Aida Nordman

Programming in C++
TNG033
Lab 3

Course goals

To use

- base-classes and derived classes,
- class inheritance,
- polymorphism, dynamic binding, and virtual functions,
- abstract classes,
- and (again), operator overloading.

Preparation

- Since operator overloading is taken once more in this lab, you need to review again Fö 9 and example 1 of lesson 2. Pay special attention to the subscript operator (**`operator[]`**).
- Review the concepts introduced in Fö 10 to 11.
- Review the examples of lesson 3.
- Read the following sections of the book.
  - Section 7 (skip 7.3.5 and 7.3.6) ,
  - Section 8 (skip 8.4.5, 8.4.9, 8.5-8.7)
    - Sections 8.4.6 and 8.4.7 of the course book discuss the subscript operator and function operator, respectively. These operators are both used in this lab.
  - Section 9.1 to 9.3, 9.6, 9.9, and 9.10.

In this lab, it may be needed to compare **`double`**s with zero. We remind you that comparisons such us (assume **`d`** is a **`double`**)

```
if (d == 0) …
```

are not correct, in fact most of the compilers generate the warning "*comparing floating point with == or != is unsafe*". Thus, you can use instead

```
if (fabs(d) < EPSILON) …
```

, where **`EPSILON`** is a small constant like

```
const double EPSILON = 1.0e-5;
```

Presenting solutions

You should demonstrate your solution orally during your redovisning lab session on **week 50**. Use of global variables is not allowed, but global constants are accepted.

Late labs can only be presented during a redovisning lab session, if time allows. This means that during the redovisning lab session in week 50, priority is given to presenting lab 3. Moreover, if you want to attend a lab session then you **must** go to the lab room that is scheduled for the lab class you signed for in the beginning of the course (see lab groups registrations if you have forgotten to which lab class you signed up for). The schedule for the labs is available from the course web site.

If you have any specific question about the exercises, then drop me an e-mail. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code to the e-mail's subject, i.e. "**TNG033: …**".

Exercise

The aim of this exercise is that you define a class hierarchy to represent certain types of expressions as described below.

Define a class **`Expression`** to represent mathematical functions of the form **`y = f(x)`**, i.e. functions of one real argument **`x`**. This class should offer the following basic functionality.

- A function, called **`isRoot`**, to test whether a given value **`x`** is a root of the function **`f`**.
- An overloaded **`operator()`** to evaluate an expression **`E`**, given a value **`d`** for variable **`x`**, i.e. **`E(d)`** returns the value of expression **`E`** when **`x`** gets the value **`d`**.
- A stream insertion operator **`operator<<`** to display an expression.
- All expressions should be clonal. A class is clonal if its instances can create copies ("*clones*") of themselves.

Define then a subclass **`Polynomial`** that represents a polynomial of a given degree, e.g. $2.2 + 4.4x + 2x^2 + 5x^3$. Your class should provide the following functionality, in addition to the basic functionality for an expression.

- A constructor that creates a polynomial from an array of **`double`**s. For example, consider the following array declaration.

  ```
  double v[3] = {2.2, 4.4, 2.0};
  ```

  Then,

  ```
  Polynomial q(2,v);
  ```

  creates the polynomial $q = 2.2 + 4.4x + 2x^2$.
- A conversion constructor to convert a real constant into a polynomial.
- A copy constructor.
- An assignment operator.
- Addition of two polynomials **`p+q`**, where **`p`** and **`q`** are polynomials. Note that statements such us

  ```
  p1+p2 = p3;
  ```

  should not compile.
- Addition of a polynomial with a real (**`double`**) value `d`, i.e. **`p+`**`d` and `d`**`+p`**, where **`p`** is a polynomial.

- A subscript operator, **`operator[ ]`**, that can be used to access the value of a polynomial's coefficient. For instance, if **`p`** is the polynomial $2.2 + 4.4x + 2x^2 + 5x^3$ then **`p[3]`** should return **`5`**. Note that statements such as

  ```
  k = p[i];
  ```
  or
  ```
  p[i] = k;
  ```

  should both compile with the expected behaviour, where **`p`** is a polynomial and **`k`** is a variable.

Finally, define another subclass (of **`Expression`**) named **`Logarithm`** that represents a logarithmic function of the form $c1 + c2 \times log_b(E)$, where E is an expression (either a polynomial or a logarithm) and $c1$, $c2$, and b are constants. You can find below some examples of logarithmic expressions that should be representable.

- $3.3log_2 x + 6$ or $2.2 \times log_2(x^2 - 1)$ or $3 \times log_{10}(log_2(x^2 - 1) + 2.2) - 1$.

`Logarithm` class should provide all the functionality described for expressions. Moreover, your class should also provide the following operations.

- A default constructor that creates the logarithm $log_2 x$.
- A constructor that given an expression E, and constants $c1$, $c2$, and b creates a logarithmic expression $c1 + c2 \times log_b(E)$.
- A copy constructor.
- An assignment operator.

Inspect the file **`test.cpp`** that already contains a **`main`** function to test your program. The tests are incremental, so that you can develop your code incrementally and test it, by commenting away those tests that correspond to member functions not yet implemented. Note that the **`main`** cannot be modified.

For each class, there should be a separate header file (**`.h`**) and a source file (**`.cpp`**).

The expected output of the program is available in the file **`out3.txt`**.

Lycka till!