Linköping University
Department of Science and Technology/ITN

Programming in C++

TNG033

Lab 2

Course goals

To use

- base-classes and derived classes,
- class inheritance,
- polymorphism, dynamic binding, and virtual functions,
- abstract classes,
- and, operator overloading: `operator[]` and `operator()`.

Preparation

You must perform the tasks listed below before the start of the lab session on week 48. In each **HA** lab session, your lab assistant has the possibility to discuss about three questions with each group. Thus, it is important that you read this lab description and do the indicated preparation steps so that you can make the best use of your time during the lab session.

To solve the exercises in this lab, you need to understand concepts such as pointers, and dynamic memory allocation and deallocation. These topics were introduced in lectures 1 to 3.

- Recall the notion of function call operator (`operator()`). This concept was introduced in lecture 7.
- Review the concepts introduced in lectures 9 and 10, e.g. inheritance, virtual functions, abstract classes.
- Download all needed files from the course website, including a file named `test.cpp` containing the `main` function to test your code. Create a project and add the downloaded files to the project.
- Implement class `Expression` whose specification is given below.
- Write the interface of class `Polynomial` (i.e. `Polynomial.h`), according to the given specification.
- Finally, implement the constructors, destructor, assignment operator, and the function call operator (i.e. `operator()`) for class `Polynomial`. Thus, before the start of your lab session on week 48, your code should pass the test phases 0, 1, 2, and 3 of the `main` function given in the file `test.cpp`.
- Write down the most important questions that you want to discuss with your lab assistant.

Presenting solutions and deadline

The exercises are compulsory and you should demonstrate your solution orally during your **RE** lab session on **week 49** After week 49, if your solution for lab 2 has not been approved then it is considered a late lab. A late lab can be presented provided there is time in a **RE** lab

session. Note that a fixed time slot is assigned to you group for presenting the lab. You can find your time slot here.

Three necessary requirements for approving your lab are given below.

- Use of global variables is not allowed, but global constants are accepted.
- The code must be readable, well-indented, and follow good programming practices.
- Your solution must not have pieces of code that are near duplicates of each other.
- The compiler must not issue warnings when compiling your code. In Visual Studio, you should set the compiler's warning level to **level4** (`\W4`).
- There are no memory leaks neither other other memory-related programming errors. We strongly suggest that you use one of the tools listed in the appendix of lab 1.
- Your code must pass all tests in the file `test.cpp`.

If you have any specific question about the exercises, then send us 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/MT2: ...`".

Comparing floating points

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 == 0.0) …
```

should be avoided, in fact several compilers generate the warning "*comparing floating point with == or != is unsafe*". The reason is that floating point numbers can only be represented in an approximate way in the computer's memory (see also the notes "Digital storage of integers and floating points " of TND012[1] course). There is no universal solution for this problem, i.e. the solution usually depends on the problem and the variables' meaning. For this lab, we suggest that you use (include the `<cmath>` library)

```
if ( std::abs(d) < EPSILON ) …
```

where `EPSILON` is a small constant like

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

[1] Use login: `TND012` and password: `TND012ht2_12`.

Exercise

The aim of this exercise is that you define a (simplified) 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 function call operator (`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 clonable. A class is clonable if its instances can create copies ("*clones*") of themselves. Thus, for any instance `o` of class `Expression`, `o.clone()` should return a (pointer) to a copy of object `o`.

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 `doubles`. 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.
- 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; //modify the coefficient of x^i
  ```

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

  A subscript operator was discussed in lecture 7 for class `Matrix`, `Matrix::operator()` – see slides 19 to 25. Recall that to access a slot of a `Matrix` object two indices are needed, line and column. Thus for the example of class `Matrix`, the `operator()` was overload

instead of `operator[]` because the latter can only have one argument, while the former can have any number of arguments.

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, this 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 the 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 there is no guarantee that your code is correct just because it passes all tests given in `test.cpp`. Thus, you should always test your code with extra examples, you come up with. However, use the original `main` function given in `test.cpp`, when presenting your lab in the **RE** session.

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 `out2.txt`.

Theory questions

While presenting your lab, you need to answer the following two questions, though other questions can also be asked. We expect that you reflect on these questions in advance.

- Investigate the reason(s) for having the member function `clone`. If the idea is to make copies of objects that are instances of class `Expression` then aren't the copy constructors in each of the derived classes of `Expression` enough?
- Consider the following piece of code. Should this code compile? Motivate your answer.

```
double v1[4] = { 2.2, 4.4, 2.0, 5.0 };
const Polynomial pp(3, v1);

pp[2] = 3.3;
```

Lycka till!