Task for OOP

deadline: Nov. 21, 2011, 11:00 Uhr

---

4. Rewrite your class `bruch` from exercise 3 as template class. Test your class with template parameters `long long int`, `short int`, `signed char` similar to the main program in exercise 3.

5. Define an STL vector with your class `bruch` as type of the elements and read its data from file input_22_a.txt. Adapt module *file_io* for that purpose, i.e., by rewriting the functions therein as template functions. The data in the file are arranged such that enumerator and denominator (Zähler, Nenner) of the first rational are followed by enumerator and denominator of the second rational, etc.. (Hint: Your class will need an input operator.)

   Use the **STL algorithms** and the methods of your container `vector` to finish the following subtasks (always check for the correct result).

   (a) Output of the read vector a via a separate function `operator<<` .

   (b) Copy vectors a onto a second vector b [STL?].

   (c) Sort vector b in increasing order [STL].

   (d) Remove all multiple elements from b [STL].

   (e) Copy all positive elements from b into a vector c [STL].

   (f) Determine the largest and the smallest element of a [STL].

   (g) Determine the largest and the smallest element of a with respect to its abolute value and remove the two elements [STL].

   (h) Sort vector c in decreasing order [STL].

   (i) Count the number of elements in a which are equal $\frac{1}{4}$ [STL].

   Hints:

   - Try the above subtasks first with `vector<int>` initialized with appropriate data. If that works than solve them with your class `bruch` (even with the template class).
   - Read the description (and examples!!) of the following algorithms/methods in www and use them whenever possible: `unique`, `copy`, `remove_copy_if`, `find`, `remove`, `remove_if`, `sort`, `max`, `max_element`, `vector<T>::erase` .
   - Add the needed comparison operators to your class `bruch` (needed for sorting and test for equality).

6. Implement the class `date` storing day, month and year that contains the following methods and functions:

   - Parameter constructor: Avoid inadmissible input parameters, e.g., month 13 or -1 as day. Choose the nearest admissible value in such cases (here 12 and 1). Don't distinguish between leap year and normal year.

- Comparison operators (<, >, ==).
- Output operator.

7. Design and implement a class `cow` with the following (private) properties/members:

   - name
   - year, month and day of birth (use class `date` !)
   - monthly milk output in liter
   - weight in kg

   Implement all necessary constructors, operators and further useful methods.

   Store also the price for meat and milk in that class (`static`) and implement a method `value` that calculates the price for the cow depending on milk output and weight. Mark methods and parameters as `const` whenever possible!

   Store the number of instances (active variable) of your class `cow` as *static member*, see www and implement a method that returns that number.

8. *Class hierarchy:* **Hands away of the keyboard!**
   Design a class hierarchy for an animal farm including e.g., layer (Legehenne), turkey, sheep, cow, dog, with one basis class for all animals from which you derive classes for bipeds (Zweibeiner) and quadruped (Vierbeiner) and so on. Each derived class should have new properties.

   This might mean also that you have to move common members and common methods from your existing class `cow` into the basis classes.

   - You always have to think over the following question during the hierarchy design: What are the common properties/members, methods of multiple classes ($\longrightarrow$ basis classes) and what are the additional properties, methods ($\longrightarrow$ derived classes).
     E.g., what are the differences between cow and sheep (layer and turkey)?
   - Each animal has a birth date and a weight.
   - At least method `wert` in EUR (value of animal depending on milk output, eggs output, weight, wool) has to be available for all animals. Implement additionally the method `PrettyPrint` that prints all data of an animal (separately from `operator<<`).
   - Take care which class in your hierarchy really have to implement a method.

   Now, you are allowed to touch the keyboard.
   Implement **now** your class hierarchy (Tutorial in English) and check your functions/methods.

9. *Abstract basis class(es):*
   Use virtual methods in your class hierarchy and, if everything runs well, use abstract basis classes.

10. *Polymorphism* Define a vector of pointers to animals (pointer at basis class) which elements are initialized for all animals from a farm (e.g., one dog, two cows, two layers, one turkey, one sheep) with useful data.

    - Write a method `Output` that only calls method `PrettyPrint` for the input parameter (via pointer at base class).
      Do you get the correct output for each animal?
      If yes then print all vector elements using this method.
    - How many cows do you have (use a method)?
    - Determine by using STL algorithms:
      (a) the oldest animal,
      (b) the most valuable animal and sell it, i.e., remove it from the farm.
      (c) Calculate the sum of the animal values and the sum of their legs.
      Hint: Use For-Loop (boring) or `accumulate`..
    - Use `operator <<` to write all data of the animals into an ASCII-file.