Dirk Praetorius,
Thomas Führer

Sommersemester 2014
12.05.2014

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Write a class `Name` which contains two members, `firstName` and `surname` of type `string`. Implement the set-method `setName` that has one string variable as input parameter, and splits the input in first name and surname automatically. Note that the input can contain multiple first names. Furthermore, write a method `printName` which prints out the whole name on the monitor. In case of multiple first names, the output should be shortened as follows: The name `Max Maxi Mustermann` should be printed out as `Max M. Mustermann`.

Aufgabe 9.2. Write a class `Hangman` that contains the methods `guessChar`, `solve`, `newString`. The class should store a string of length n which has to be guessed. The method `guessChar` allows the user to guess a single character in the string. In case that the string contains the character, the method `guessChar` should return the index resp. the indices of the the character in the string. In case that the string does not contain the character, an appropriate message should be printed out. The user loses, if he is not able to find the correct string after 8 tries. Additionally, write a method `newString` to start the game with a new word, and a method `solve` which allows to solve it. Moreover, write a main program to check if your implementation is correct.

Aufgabe 9.3. Write a class `Deposit` with members `accountNumber`, `assets`, and `ratePerCent`. Moreover, implement `set` and `get` methods for the members `accountNumber`, `assets`. To change the assets, write a method `drawMoney` and `placeOnDeposit`. Note that with this deposit you are not allowed to draw more money than is given, i.e., the member `assets` must be positive. The rate per cent as well as the account number must also be positive. Finally, implement the method `calculateAssets`.

Aufgabe 9.4. Write a class `Client` that stores a list of deposits. Use the container `vector`! Furthermore, the class should contain an object of the class `Name` from Exercise 9.1. Implement methods for adding and deleting deposits. Moreover, write a function that computes the assets of all deposits. Think of other useful functions.

Aufgabe 9.5. Write a class `Stopwatch` that simulates a stopwatch. The stopwatch consists of two buttons: If the first button is pressed, then the time measurement starts. If the button is pressed again, then the time measurement stops. The second button is used to reset the time to zero. To realize this situation, implement the methods `pushButtonStartStop` (first button) and `pushButtonReset`. Implement another method that prints out the time formatted in the style `hh:mm:ss.xx`, e.g., if the measured time is two minutes, then the output should be `00:02:00.00`.
Hint: Use the data-type `clock_t` and the function `clock()` from the library `time.h`. It makes sense to use a variable `isRunning` of type `bool`. If the first button is pressed, then this variable is either set to `true` or `false`.

Aufgabe 9.6. Write a class `University`. This class should contain the members `numStudents`, `city`, and `name` as well as the methods `graduate`, and `newStudent`. If the method `graduate` is called, the number of students gets decreased by one, whereas if `newStudent` is called, the number of students increases by one. All data-members should be declared as `private`! Therefore, you have to implement `get` and `set` methods.

Aufgabe 9.7. Implement the `get` and `set` methods of the class

```
class Fraction {
   long numerator;
   unsigned long denominator;
public:
   Fraction();
   Fraction(long numerator, unsigned long denominator);
   setNumerator(long z);
   setDenominator(unsigned long n);
   double getValue();
};
```

The method `getValue` should return the floating-point value of the fraction. Take care of the fact, that the denominator has to be nonzero. Additionally, implement the constructor `Fraction()` that sets the numerator to 0 and the denominator to 1.

Aufgabe 9.8. Extend the class `Fraction` by the `public` method `void reduce()` that determines the reduced form of the fraction `numerator/denominator`. Use the *euclidean division algorithm.* Moreover, implement the method `setValue(string value)` that converts an arbitrary number, given as a string, into a fraction. For the implementation you can proceed as follows: First, find the decimal-point in the string and count the number of positions after the decimal-point. Then, erase the decimal-point from the string. The string now represents a natural number and can be converted into an `int` variable by use of the function `atoi`. This number is used for the numerator. Then, the denominator is set to 10^p, where $p \in \mathbb{N}$ is the number of positions after the decimal-point. Finally, call the method `reduce()`.
Hint: The method `find` of the class `string` allows you to find a specific character in the string, e.g., `int pos = value.find('.')` returns the position of the decimal-point in the string `value`. The call `value.erase(pos,k)`, erases `k` characters after the position `pos` in the string `value`. The function `atoi` from the standard library `cstdlib` converts a given string (in C-style) to an `int` variable. To get the string as `char *`, you can use the method `c_str()` of class `string`.