Institut für Automatisierungstechnik

C++ Basics and Applications in technical Systems

Exercise II (Lecture 4-6) in WiSe 2015/2016

Further information and this exercise as download on:
http://www.elearning.uni-bremen.de

Supervisors:
Adrian Leu - Santiago Focke - Björn Mindermann
{aleu, sfocke, bmindermann}@iat.uni-bremen.de

General comment: Always make sure to inform the user what he should do (use a menu with instructions) and to check if the user entered the correct data type, so that the program does not crash! (e.g. character, number).

1. Create a module for image processing functions. Within this module implement four functions that are able to invert a passed bmp image file (use the file fruit.bmp, found in elearning, to test your module). The four functions differ only in the way the image is passed to them. Use the following different ways of passing the image: (15)

   - Pass the image as `std::vector<char>` to the function, invert it and return it back to the main.
   - Pass the image as reference of `std::vector<char>` to the function and invert it.
   - Pass the image as pointer to `std::vector<char>` to the function and invert it.
   - Pass the image as character array to the function and invert it.

   Implement a main application that is capable of handling command line arguments in the form,

   ```
   int main(int argc, char ** argv)
   {
     // ...
   }
   ```

   so that one can call your application like:

   ```
   ImageConvert InputImage.bmp ResultImage.bmp USE_POINTER
   ```

The first argument is the source image file, the second is the target image file and the third is used to specify which function from the module has to be used to process the image.

To invert the image you have to calculate for each value of the image data the new value in the following form:

```
resultValue = 255 - sourceValue
```

Within your main application do some time measurement to get the duration of the converting process. For this purpose you can use:

```
#include <time.h>

double dStart = clock();

// Image processing ...

double dDuration = clock() - dStart;
std::cout << dDuration / CLOCKS_PER_SEC << " sec." << std::endl;
```

Test your program and compare the time consumption for the different functions from your module.

Notes:

- A BMP image consists of a 54 bytes long header followed by the image data.
- Calculate the new values only for the image data and not for the header. Otherwise the image is not readable anymore.
- Read the input image file in binary mode! E.g.:
  `ifstream inputFile("InFile.bmp", ios::in | ios::binary);`
- Write the output image file in binary mode! E.g.:
  `ofstream outputFile("OutFile.bmp", ios::out | ios::binary);`

2. Making coffee seems to be very simple since (almost) everybody knows how to do it. However, to train thinking in an object-oriented way, a coffee machine class shall be designed that provides some smart behavior. Probably, after finishing this exercise, we become aware that the coffee cooking procedure is not as simple as initially assumed. (24)

   Create a class `IntelligentCoffeeMachine` that fulfills the following requirements:

   (1) **R01 Prerequisites:** For simplification we assume that we have a coffee machine with fixed water and coffee reservoir and the coffee jug is not used for water insertion, but some external jug.

   (2) **R02 User interaction:** The user (coffee drinker) shall be able to interact with the coffee machine (IntelligentCoffeeMachine class) as follows:
   - Open lid
   - Insert filter
   - Insert coffee
   - Insert water
   - Close lid
   - Start cooking
   - Take coffee
   - Switch off

(3) **R03 Intelligence in general:** The „intelligence“ of the coffee machine prevents wrong actions. In general the internal states have to be checked before action execution (E.g. Opening the lid is only possible in case the lid is not already open). In case of wrong user interaction the user is informed with a respective text message. See subsequent requirements for special status requirements that depend on each other.

(4) **R04 Intelligent coffee insertion:** Coffee can only be inserted in case the lid is open and the filter is inserted. Respect the maximum content of the coffee reservoir (class related constant).

(5) **R05 Intelligent water insertion:** Water can only be inserted in case the lid is open. Respect the maximum content of the water reservoir (class related constant).

(6) **R06 Intelligent cooking:** Coffee can only be cooked when minimum amount of water and coffee are in the reservoirs. When the user switches the coffee machine on to start cooking, a power LED is lit (boolean attribute of the class).

Since we do not want to have parallel threads here, the coffee cooking is simulated by a timer or a loop (blocking behavior). Visualization of the cooking progress could be helpful. After finishing of the cooking procedure the coffee jug is filled and the water reservoir is empty.

(7) **R07 Intelligent coffee taking:** The user is only allowed to take the coffee when the cooking procedure is finished. This means the coffee jug is filled and the water reservoir is empty. Before taking the coffee he must switch off the coffee machine to assure no danger of a coffee machine still switched on. (We assume that the user does not want to keep the coffee heated – and probably drinks all the coffee at once, or has some friends/colleagues to share the coffee with ...)

(8) **R08 Commenting:** Comment all entities of your class using C++ style comments (`//` or `/**/`).

(9) **R09 Intelligent clean up:** After the coffee has been cooked, the old filter must be taken out and a new one inserted before filling in new coffee.

Write a test application with a menu for using the coffee machine.

3. Define a class **employee** with the features name and salary which could not be accessed from outside the class. Create a constructor, by which those features can be initialized during the object definition. The class should have two methods: *works* which has no return value, no arguments to be passed and prints name and salary on the screen. The second method is *reports* and does not have a return value and arguments to be passed, too. The second method is empty. (16)

   Define a class **manager** and derive it from the class employee. Define a constructor by which a name can be assigned during the object definition. The salary for every manger is 10.000 EUR per month. This should also be set during initialization. The manager class should also have a *reports* method which prints a message on the screen that the manager reports to the CEO.

   Define a class **department** which has the features department name and meeting and a *reports* method without parameters to be passed or returned. In the feature meeting a weekday should be stored later.

   Define a class **marketing** and derive it from the class department. The marketing department always meets on Tuesdays and reports to the executive department.

   Define a class **chief of marketing** which is derived from the classes manager and marketing. Create also a special constructor to give a name during object definition. Create an object of type chief of marketing and call for it in the main function the *reports* method to print on the screen to whom the marketing department reports.

HowTo: Send in the Exercises

(Exercises are only accepted if they meet the following conditions)

- Consulting with colleagues is allowed, but copying is **not** permitted! Solutions will be accepted no later than the announced deadline.
- Special formatting requirements: Use spaces and tabs to align the lines of code within command blocks and use comments to explain what you did. Non-readable or not documented code will be severily graded down! If you use Qt, you can select all the code with Ctrl+A and then conveniently press Ctrl+I for automatic indentation. For more details about our guidelines see **IAT programming guidelines**.
- zip/rar the project-folder(s) and send everything in one file via Email or hand it in via usb-stick. Before compressing, delete the object-files (*.o) and the debug/release folders that contain the executable files. **Additionally a paper printout has to be delivered to the supervisors**.
- Preferably write your programs with Qt Creator. If you do not use Creator, you have to provide the source code (again delete object-files and executbale files) together with a working Makefile. If you use CASE-tools (like Rhapsody or Bouml) send in a full set of model-files **and** the generated C++ source code.