**These assignments were first used in the C++ course starting in May 2014.**

Build your programs in a neat object oriented way as explained in the lessons.

Hand in your assignments after you've finished and tested them.

Not everyone will make all assignments. Especially after the first few lessons, assignments are given on an individual basis, depending on your skills.

Each week assignments will be added to this document.

**Assignment 1**

Define a class Age. It has a field (instance variable) of type real or integer to store your age and it has member functions that ask for your age and give a reply, depending upon which age was entered. Use an if...else construction to make your class respond differently to different ages.

Instantiate and use your Age class in a program containing the main function. Call this program e.g. main.cpp.

Your class age is declared in age.h and its member functions are defined in age.cpp. Include age.h in both age.cpp and main.cpp to achieve inter-module type checking as explained in the lessons.

**Assignment 2**

Make a C++ program that displays the multiplication tables into a console window, e.g. like:

*1 x 1 = 1 1 x 2 = 2 ... ... 1 x 10 = 10*

*2 x 1 = 2 2 x 2 = 4 ...*

*... ... ...*

*10 x 1 = 10 10 x 10 = 100*

N.B. It is ok to print the tables below each other rather than next to each other.

**Assignment 3**

Make a C++ program that children can use to practice the multiplication tables from 1 to 10.

The program will ask questions in random order like

*How much is 7 x 8?*

The questions are repeated in random order until each question was answered correctly for 3 consecutive times.

N.B. Don't ask the same question again and again until it is answered properly 3 times. Keep asking questions in random order.

Use a Table class, with a constructor receiving one parameter, indicating for which multiplication table the class is instantiated. So *new Table (3)* will instantiate the multiplication table of 3. Store your tables into an array indexed from 0..9 Each Table has a score array of ten elements, counting the number of consecutive correct answers. Each time a wrong answer is given, the score is reset to 0.

So if in case:

*table = new Table (3)*

table.score [5] belongs to question:

*How much is 6 x 3*

N.B. Array indexing starts at 0

If the answer is right, the score is incremented, if it is wrong, the score is reset to 0.

Any questions with scores >= 3 are no longer asked.

If all scores are >= 3, the program stops. So any time a score reaches 3, you increment a counter until it reaches 10. This is far more efficient that checking all your scores after each correct answer.

**Assignment 4**

Make a program with the following class hierarchy:

A Shower is an object that shows itself, e.g. by printing a random message, taken from an array of messages.

A Beeper is an object that produces a beep of random length and random pitch.

An Annoyer does it both, by inheriting from classes Shower and Beeper. Preferably a Beeper beeps and shows himself simultaneously! If that is problematic, it should instead beep for the same length and pitch before and after it has shown its message.

All three classes inherit from an abstract base class called Actor. Actor has a function doYourThing.

In your main function, define an array with pointers to Actor, so Actor \*. Randomly allocate Beepers, Showers and Annoyers and "hang" them behind the pointers in the array. Then traverse this array in sequence order, having each Actor do its thing.

Purpose is to practice with pointers and virtual functions. Don't create memory leaks, so deallocate your Actors at the end of your program.

**Assignment 5**

This assignment is about operator overloading.

Step 1. Make a Matrix class, objects of which are matrices of dimension m x n. Give your class a constructor Matrix (int nRows, int nColumns), that allocates a one-dimensional array field of nRows x nColumns doubles. Initialize this array field to all zero's.

Test this step.

Step 2. Give your Matrix class the following methods:

Method setElement (int iRow, int iColumn, double value) will store value in the array element with index iRow x nColumns + iColumn.

Method double getElement (int iRow, int iColumn) will retrieve value from the array element with index iRow x nColumns + iColumn.

Matrix setMatrix (Matrix m) will copy the m's value array to \*this.

Method print will print out the array.

Test this step.

Step 3. Create the following global functions:

Matrix add (Matrix m0, Matrix m1) returns m0 + m1.

Matrix multiply (Matrix m0, Matrix m1) returns m0 \* m1.

Test this step.

Step 4. Replace the functions of step 3 by overloaded global operator functions:

Function add becomes function operator+.

Function multiply becomes function operator\*.

Test this step.

Step 5. Replace the functions of step 2 by overloaded operator methods:

Function setElement and getElement are combined in double &operator() (int iRow, int iColumn).

Function setMatrix is replaced by Matrix &operator= (Matrix m).

Test this step.

Step 6. (bonus) Overload the constructors and, if needed, other functions of Matrix to make the following possible:

Matrix m0 = 1;

Matrix m1 (3, 4);

m1 (0, 0) = 5.3; // etc. (fill matrix)

Matrix m2 = m0 \* m1;

m2 = 2 \* m0;

m2 = m0 \* 3;

Test this step.