Game Engine Design

Assignment 1 – *"C++ Basics!"*

For this assignment you will implement some basic C++ stuff which will be useful for upcoming assignments.
The slides will contain a more detailed explanation and hints for some parts of the assignment!
*Note that this assignment **will not be graded**, but you will receive feedback from your tutor nevertheless if you commit your solution on time.*

Hello World

- SVN-Checkout the "`Hello World`" Solution from assignment 0 and extend your `HelloWorld.cpp` file in such a way that "`Hello World`" is printed on the console. Make use of the standard output (`std::cout`) and don't forget to `#include <iostream>`.
- Hint: You can prevent the console from closing by adding `system("pause");` before the final `return 0;`
- Now perform an SVN-Commit which concludes this part of the assignment.

Smoothing Values in a 2D Array

Let us now do something more useful. Your goal is to smooth scalar values (ie. real numbers) which are given in a 2D array. This will become handy when we generate landscapes in the next assignment.

- Add a new project called "`2DArraySmoothing`" to your existing solution (File -> Add -> New project…). Like in the previous assignment choose again "Win32 Console Application" and uncheck the "Precompiled header" checkbox. Now select this project as StartUp project (Project -> Set as StartUp Project). And, of course, SVN-Add and SVN-Commit your new files following the same procedure as in the previous assignment.
- To create a 2D array first choose some integers for its dimensions (`width` and `height`) and then create a new `float`-array of size `width * height`.
- Now the value at position `(x,y)` is stored at index `x + y * width` in this array. You can use the following macro for a more convenient access to the array position (x, y); just copy it into your cpp file.

```
// Access a 2D array of width w at position x / y
#define IDX(x, y, w) ((x) + (y) * (w))
```

- Fill your array with random values between `0.0f` and `1.0f`.
  - Hint: Use the `rand()` function and the `RAND_MAX` constant to generate these values (For this `#include <cstdlib>`). Don't forget to seed before the first call!
- Add a function `printArray` which accepts a pointer to float and two integers for the array dimensions. This function should print the array contents to the standard output.

- Write a function `smoothArray` which accepts a pointer to float and two integers for its dimensions. This function should perform a smoothing operation in the following way: Each value should be replaced by the average of itself and the surrounding values (a total of 9 values except at the borders).
- Hints:
  - You can't manipulate the values in-place, ie. in the same array, since you always need the initial (non-smoothed) surrounding values for calculating the average. Create an additional temporary array of the same size for storing the smoothed values and later write its content back to the original array.
  - Take special care of the borders such that you do not access values outside the array.
  - Use the debugger to check if your algorithm works as desired.
- Print your array to the console, then call the `smoothArray` function and finally print the smoothed array again. Do the average calculation for some randomly selected samples by hand to verify that everything works as expected.
- Don't forget to release dynamically allocated memory, ie. for every `new` there must exist exactly one `delete` call.
- Finally SVN-Commit everything.

Sorting a Vector of Integers

We now introduce the concept of vectors which are part of the standard library. Vectors represent arrays and provide useful additional functionality.

- Add a new project called "`VectorSort`" to your solution (you know how), set it as StartUp project and once again perform an SVN-Add and SVN-Commit.
- Make sure to `#include <vector>` in the main cpp-file and start with creating a new `std::vector<int>` object in the main function.
- Now read integers from the standard input (`std::cin`) until the user enters `0` and add them immediately to your vector object (`#include <iostream>`).
- Then sort the vector in a **descending** order by calling the `std::sort` function (`#include <algorithm>`).
- Hints:
  - You also need to define a comparison function.
  - You find an explanation and an example here: http://www.cplusplus.com/reference/algorithm/sort/
- Print the sorted vector to the console by iterating over the vector and again making use of the standard output.
- Again, SVN-Commit everything.

Configuration Parser

For the last part of this assignment you'll implement a program which parses a configuration file and stores these values in the memory. For the upcoming assignments you can reuse this code.

- Add a new project called "`ConfigurationParser`" to your solution, set it as StartUp project and then add a class "`ConfigParser`" to this project (Project -> Add Class ->

C++ Class. Then click on Add and enter `ConfigParser` as the class name). Now it's time for SVN-Adding and committing, as you might already know.

- In the `public` part of the class header, declare a `struct Color` which consists of three `float`-values `r`, `g`, `b`.
- Add the following as private members to your class (you need to `#include <string>`):

```
float           spinning;
float           spinSpeed;

Color           backgroundColor;

std::string     terrainPath;

float           terrainWidth;
float           terrainDepth;
float           terrainHeight;
```

Hint: You need to put the `public`-part of the class before the `private`-part, since `Color` needs to be declared before you can use it.

- In the external folder (SVN) you find the configuration file `game.cfg`. Copy this file to your project folder and open it with a text editor of your choice. As you can see each line contains a pair consisting of a variable name and a value.
- Now implement a public function called `load` in your class `ConfigParser` which accepts a string containing the filename of the configuration file. When called it should open the specified config file and parse its values in the following way:
    - Open the file by using `std::ifstream` (`#include <fstream>`) .
        - Hint: Use the function `is_open` to determine whether the file was successfully opened http://www.cplusplus.com/reference/fstream/ifstream/
    - Read the first word of each line into an `std::string` using the operator `>>`. Compare this value to a list of known parameters and read the arguments into the respective member variables.
        - Hint: You may need to read more than a single value, e.g. for `backgroundColor`. You can directly chain the operator `>>` (see the slides for more details)!
        - Hint: It might be useful to add an error message when an unknown parameter is encountered.
    - After reading the whole file, don't forget to close it by calling the `close` function.
- Implement for each private class member a public 'get'-function which returns its value.
- To check if everything is working as expected create a `ConfigParser` object in your main function, then call the `load` function and finally print each value to the console by calling the 'get'-functions.
- SVN-Commit everything you have done.

If you face any difficulties, please use the Q&A Forum at https://qage.in.tum.de/ or ask your tutor.

The working solution must be committed till **May 1st, 10:00**. Otherwise it might not be revised by your tutor. If anything is not working, explain yourself in the “readme.txt” file within your “solution” directory.